



مركز البحوث

إدارة وتطوير مشاريع البرمجيات

تأليف:

د. عبدالوهاب كامل السماك

بسم الله الرحمن الرحيم



مركز البحوث

إدارة وتطوير مشاريع البرمجيات

تأليف

د. عبدالوهاب كامل السماك

١٤٢٩هـ - ٢٠٠٨م

بطاقة الفهرسة

⑦ معهد الإدارة العامة، ١٤٢٩هـ.

فهرسة مكتبة الملك فهد الوطنية أثناء النشر.

السماك، عبد الوهاب كامل

إدارة وتطوير مشاريع البرمجيات/ عبد الوهاب كامل السماك

الرياض، ١٤٢٩هـ

٣٢٨ صفحة ١٧ × ٢٤ سم.

ردمك: ٥-١٦٩-١٤-٩٩٦٠-٩٧٨

١- البرمجة (حواسيب) ٢- تخزين واسترجاع البرمجيات

أ. العنوان

١٤٢٩/١٠٩٣

ديوى ٠٠٥,٣٦٩

رقم الإيداع: ١٤٢٩/١٠٩٣

ردمك: ٥-١٦٩-١٤-٩٩٦٠-٩٧٨

إهداء

إلى والدتي وروح أبي

إلى زوجتي وأولادي

إلى أساتذتي وزملائي

وإلى طلاب العلم في الوطن العربي

قائمة المحتويات

رقم الصفحة	الموضوع
١٩	مقدمة
٢١	الجزء الأول: مشاريع البرمجيات Software Projects
٢٥	الفصل الأول: دورة حياة تطوير البرمجيات Software Development Lifecycle
٢٦	١-١ أساسيات هندسة البرمجيات Software Engineering Fundamentals
٢٧	١-١-١ النظام البرمجي Software System
٢٨	٢-١-١ العملية البرمجية Software Process
٢٩	٣-١-١ هندسة البرمجيات Software Engineering
٣١	٤-١-١ نمذجة البرمجيات Software Modeling
٣٣	٢-١ مراحل دورة حياة تطوير البرمجيات Lifecycle Phases
٣٣	١-٢-١ تحليل المتطلبات Requirements Analysis
٣٦	٢-٢-١ تصميم النظام System Design
٣٧	٢-٢-١ التطبيق Implementation
٣٩	٤-٢-١ التكامل والنشر Integration and Deployment
٤١	٥-٢-١ التشغيل والصيانة Operation and Maintenance
٤٢	٣-١ طرق تطوير البرمجيات Software Development Methods
٤٣	١-٣-١ طريقة الشلال Waterfall Method
٤٥	٢-٣-١ طريقة التكرار مع الزيادة Iterative with Increment
٤٧	١-٢-٣-١ النموذج الحلزوني Spiral Model
٤٩	٢-٢-٣-١ العملية العقلانية الموحدة Rational Unified Process (RUP)
٥٢	٣-٢-٣-١ التشبيد المبني على النموذج Model Driven Architecture (MDA)
٥٥	الفصل الثاني: نمذجة البرمجيات Software Modeling
٥٦	١-٢ النمذجة الهيكلية Structured Modeling
٥٧	١-١-٢ مخططات تدفق البيانات Data Flow Diagrams (DFDs)
٥٩	١-١-٢-٢ مخطط السياق Context Diagram

رقم الصفحة	الموضوع
٦٠	٢-١-١-٢ مخطط المنظور العام Overview DFD
٦٣	٢-١-١-٢ المخططات التفصيلية Detailed DFDs
٦٦	٢-١-٢ وصف منطق العمليات Process Logic Description
٦٦	١-٢-١-٢ الإنجليزية الهيكلية Structured English
٦٨	٢-٢-١-٢ جداول القرارات Decision Tables
٦٩	٢-٢-١-٢ شجرة القرارات Decision Tree
٧٠	٢-١-٢ مخطط العلاقة بين الكائنات Entity Relationship Diagram (ERD)
٧٠	١-٢-١-٢ كائنات النظام System Entities
٧٢	٢-٣-١-٢ العلاقة بين الكائنات Entities Relationship
٧٣	١-٢-٣-١-٢ تعددية العلاقة Cardinality
٧٣	٢-٢-٣-١-٢ درجة العلاقة Degree of Relationship
٧٨	٢-٢ النمذجة الموجهة بالكائنات Object-Oriented Modeling
٨١	١-٢-٢ مقدمة عن لغة UML
٨٣	٢-٢-٢ نمذجة النظم باستخدام لغة UML
٨٤	٢-٢-٢ مكونات لغة UML
٨٥	٤-٢-٢ مخطط الأصناف Class Diagram
٩٣	٥-٢-٢ مخطط حالات الاستخدام Use Case Diagram
١٠٢	٦-٢-٢ مخطط التابع Sequence Diagram
١٠٣	٧-٢-٢ مخطط الحالة State Diagram
١٠٧	٨-٢-٢ مخطط النشاط Activity Diagram
١٠٨	٩-٢-٢ مخطط المكونات Component Diagram
١١٢	١٠-٢-٢ مخطط النشر Deployment Diagram
١١٥	١١-٢-٢ مخطط الحزم Packages Diagram
١٢١	الفصل الثالث: أدوات هندسة البرمجيات Software Engineering Tools
١٢١	١-٢ أدوات إدارة مشاريع البرمجيات Software Project Management Tools
١٢٢	١-١-٢ جدول المشروع Project Scheduling
١٢٣	١-١-٢-٢ مخطط جانت Gantt Chart
١٢٨	٢-١-٢-٢ مخطط بيرت PERT Chart
١٢١	٢-١-٢-٢ طريقة المسار الحرج Critical Path Method

الصفحة	الموضوع	رقم
١٣٢	Performance Management إدارة الأداء	٢-١-٣
١٣٣	Collaboration Management إدارة التعاون	٣-١-٣
١٣٤	Cost Management إدارة التكاليف	٤-١-٣
١٣٥	Risk Management إدارة المخاطر	٥-١-٣
١٣٦	System Modeling Tools أدوات نمذجة النظام	٢-٢-٣
١٣٧	Managing Requirements إدارة المتطلبات	١-٢-٣
١٣٩	Visual UML Modeling النمذجة المرئية	٢-٢-٣
١٣٩	Report Generation توليد التقارير	٣-٢-٣
١٤١	Database Modeling نمذجة قواعد البيانات	٤-٢-٣
١٤٢	Integrated Development Environment (IDE) بيئة التطوير المتكاملة	٣-٢-٣
١٤٢	Writing, Executing, and Debugging Programs كتابة وتنفيذ وتصحيح البرامج	١-٣-٣
١٤٣	Integration with Software Models التكامل مع نمذجة البرمجيات	٢-٣-٣
١٤٣	Enterprise Application Development تطوير تطبيقات الأعمال	٣-٣-٣
١٤٤	Integration with Business Components التكامل مع مكونات نظام الأعمال	٤-٣-٣
١٤٦	CASE Tools أدوات هندسة البرمجيات المدعمة بالحاسب	٤-٣-٣
١٥١	Software Project Management إدارة مشاريع البرمجيات	الفصل الرابع:
١٥٤	Project Plan Development تطوير خطة المشروع	١-٤
١٥٦	Project Scheduling جدولة المشروع	٢-٤
١٥٦	Tasks and Deliverables المهام والمستلزمات	١-٢-٤
١٥٨	Task Scheduling جدولة المهام	٢-٢-٤
١٦٠	Resources Calendars تقويمات الموارد	٣-٢-٤
١٦١	Effort Scheduling جدولة المجهود	٤-٢-٤
١٦٢	Project Budget Estimation تقدير ميزانية المشروع	٣-٤
١٦٢	Schedule-Driven Budget Estimation تقدير الميزانية حسب الجدولة	١-٣-٤
١٦٤	Algorithmic Budget Estimation تقدير الميزانية حسابياً	٢-٣-٤
١٦٨	Tracking Project Progress متابعة تقدم المشروع	٤-٤
١٦٩	Tracking the Project Schedule متابعة جدول المشروع	١-٤-٤
١٧٠	Tracking the Budget متابعة الميزانية	٢-٤-٤

الموضوع	رقم الصفحة
٥-٤ إدارة الموارد البشرية Human Resources Management	١٧٢
١-٥-٤ اكتساب وتحفيز الأفراد Acquiring and Motivating People	١٧٣
٢-٥-٤ تكوين فريق العمل Team Work Formation	١٧٣
٣-٥-٤ اتصالات المشروع Project Communications	١٧٥
٤-٥-٤ تنمية فريق العمل Team Work Development	١٧٨
٦-٤ إدارة المخاطر Risk Management	١٧٩
١-٦-٤ تحديد المخاطر Risk Identification	١٧٩
٢-٦-٤ تقييم المخاطر Risk assessment	١٨١
٣-٦-٤ معالجة المخاطر Risk Handling	١٨١
٧-٤ إدارة الجودة Quality Management	١٨٣
١-٧-٤ جودة البرمجيات Software Quality	١٨٤
٢-٧-٤ مراقبة الجودة Quality Control	١٨٥
٣-٧-٤ تأمين الجودة Quality Assurance	١٨٧
٨-٤ إدارة التشكيل والتغيير Change and Configuration Management	١٨٨
١-٨-٤ التغيير فى المتطلبات Requirements Change	١٨٨
٢-٨-٤ إصدارات المنتج Product Versions	١٨٩
٣-٨-٤ العيوب والتحسينات Defects and Enhancements	١٩٠
الجزء الثانى: تطوير مشاريع البرمجيات Software Projects Development	١٩٣
الفصل الخامس: تحديد المتطلبات Requirements Determination	١٩٧
١-٥ نموذج كائنات نظام الأعمال Business Object Model	١٩٧
١-١-٥ نظام الأعمال The Business	١٩٨
٢-١-٥ مخطط السياق لنظام الأعمال Business Context Diagram	١٩٩
٣-١-٥ مخطط حالات الاستخدام لنظام الأعمال Business Use Case Diagram	٢٠٠
٤-١-٥ مخطط الأصناف لنظام الأعمال Business Class Diagram	٢٠١
٢-٥ نموذج كائنات المجال Domain Object Model	٢٠٢
١-٢-٥ المجال Domain	٢٠٣
٢-٢-٥ مخطط حالات الاستخدام للمجال Domain Use Cases Diagram	٢٠٣
٣-٢-٥ مخطط الأصناف للمجال Domain Class Diagram	٢٠٤

الموضوع	رقم الصفحة
Case Study حالة دراسية	٢٠٥
Use Case Model استخدام	٢٠٦
Use Case Document وثيقة حالة الاستخدام	٢٠٧
Classes الأصناف	٢١١
الفصل السادس: التصميم المعماري للنظام Architectural System Design	٢١٣
Architectural Layers and Dependency Managemen الطبقات المعمارية وإدارة التبعية	٢١٣
Architectural Modules الوحدات البرمجية المعمارية	٢١٣
Package Dependencies تبعية الحزم البرمجية	٢١٤
Layer Dependencies طبقة التبعية	٢١٥
Class Dependencies تبعية الصنف	٢١٦
Inheritance Dependencies تبعية التوريث	٢١٧
Method Dependencies تبعية الطرق	٢٢٠
Interfaces الواجهات	٢٢١
Event Processing معالجة الأحداث	٢٢٣
Architectural Frameworks أطر معمارية	٢٢٤
Model-View-Controller (mvc) إطار نموذج-منظر-متحكم	٢٢٤
Presentation-Control-Mediator-Entity-Foundation إطار عرض-تحكم-وسيط-كائن-أساس	٢٢٥
Architectural Patterns أنماط معمارية	٢٢٧
Façade الواجهة	٢٢٧
Abstract Factory المصنع المجرد	٢٢٨
Chain of Responsibilities سلسلة المسؤوليات	٢٢٨
Observer نمط المراقب	٢٢٩
Mediator نمط الوسيط	٢٣٠
الفصل السابع: تصميم قواعد البيانات Database Design	٢٣٣
Relational Databases قواعد البيانات المرتبطة	٢٣٤
Table الجدول	٢٣٤
Referential Integrity التكامل المرجعي	٢٣٥
Conceptual Model النموذج المفاهيمي	٢٣٦

الموضوع	رقم الصفحة
٤-١-٧ Logical Model النموذج المنطقي	٢٣٧
٥-١-٧ Implementing Business Rules تطبيق قواعد العمل	٢٣٨
٦-١-٧ Programming Database Applications برمجة تطبيقات قواعد البيانات	٢٣٩
٢-٧ Object Databases قواعد البيانات الشيئية	٢٤١
٣-٧ Object-Relational databases المرتبطة - قواعد البيانات الشيئية	٢٤٣
٤-٧ Database Design تصميم قواعد البيانات	٢٤٧
١-٤-٧ Database Model نموذج قواعد البيانات	٢٤٨
٢-٤-٧ Creating Database Schema إنشاء مخطط قواعد البيانات	٢٤٩
٣-٤-٧ Sample Database Content محتوى عينة قواعد بيانات	٢٥١
الفصل الثامن: تصميم الأصناف والتفاعلات Classes and Interactions Design	٢٥٢
١-٨ إيجاد الأصناف من حالات الاستخدام Finding Classes from Use Cases	٢٥٣
٢-٨ تصميم الأصناف Classes Design	٢٥٦
٣-٨ مخطط الكائنات Object Diagram	٢٦١
٤-٨ التفاعلات Interactions	٢٦١
١-٤-٨ مخطط التابع Sequence Diagram	٢٦٢
٢-٤-٨ مخطط الاتصال Communication Diagram	٢٦٢
٣-٤-٨ مخطط التفاعل Interaction Diagram	٢٦٤
٥-٨ تطبيق على التفاعلات Interactions Application	٢٦٥
١-٥-٨ التفاعل "Login"	٢٦٦
٢-٥-٨ التفاعل "Exit"	٢٦٦
٣-٥-٨ التفاعل "View Unsent Messages"	٢٦٨
٤-٥-٨ التفاعل "Display Message Text"	٢٦٩
٥-٥-٨ التفاعل "Email Message"	٢٧٠
٦-٥-٨ التفاعل "Incorrect User Name or Password"	٢٧٠
٧-٥-٨ التفاعل "Incorrect Option"	٢٧٣
٨-٥-٨ التفاعل "Too Many Messages"	٢٧٤
٩-٥-٨ التفاعل "Email Could not be Sent"	٢٧٥

الموضوع	رقم الصفحة
الفصل التاسع: البرمجة والاختبار Programming and Testing	٢٧٧
١-٩ فكرة عامة عن لغة جافا Overview of JAVA Language	٢٧٧
١-١-٩ الأصناف Classes	٢٧٨
٢-١-٩ ارتباط الأصناف Classes Associations	٢٨٠
٢-١-٩ الاتصال بقواعد البيانات Database Connections	٢٨٥
٢-٩ التطوير المبني على الاختبار Test-Driven Development	٢٨٨
٣-٩ اختبار البرمجيات Software Testing	٢٩٣
الفصل العاشر: تصميم واجهات المستخدم User Interface Design	٢٩٧
١-١٠ إرشادات لتصميم واجهة المستخدم User Interface Design Guidelines	٢٩٨
٢-١٠ مكونات واجهة المستخدم User Interface Components	٣٠١
١-٢-١٠ الحاويات Containers	٣٠١
٢-٢-١٠ القوائم Menus	٣٠٣
٣-٢-١٠ أدوات التحكم Controls	٣٠٤
٣-١٠ معالجة أحداث واجهات المستخدم User Interface Event Handling	٣٠٥
٤-١٠ الأنماط وواجهات المستخدم Patterns and User Interfaces	٣٠٦
١-٤-١٠ نمط المراقب Observer	٣٠٦
٢-٤-١٠ نمط النقاش Decorator	٣٠٧
٣-٤-١٠ نمط سلسلة المسؤولية Chain of Responsibility	٣٠٧
٤-٤-١٠ نمط الأمر Command	٣٠٧
٥-١٠ واجهات المستخدم على الويب Web-Based User Interface	٣٠٨
٦-١٠ تمكين التقنيات لمستخدمي الويب Enabling Technologies for Web Clients	٣٠٨
١-٦-١٠ لغة HTML	٣٠٩
٢-٦-١٠ لغات إعداد صفحات الويب Scripting Languages	٣١١
٣-٦-١٠ البرمجيات Applets	٣١٢
٧-١٠ تمكين التقنيات ل خادم الويب Enabling Technologies for Web Server	٣١٥
١-٧-١٠ تقنية Servlet	٣١٥
٢-٧-١٠ تقنية JSP	٣١٧
٨-١٠ تطبيق على واجهات المستخدم User Interface Application	٣١٩
المراجع	٣٢٣

قائمة الأشكال

الشكل	رقم الصفحة
الفصل الأول:	
شكل (١-١) دورة حياة البرمجيات	٢٥
شكل (٢-١) مخطط بيئة نظم الأعمال	٢٧
شكل (٣-١) العملية البرمجية للمستويات الإدارية المختلفة	٢٨
شكل (٤-١) عملية تحديد المتطلبات	٣٤
شكل (٥-١) نموذج عام لعمليات التصميم	٣٨
شكل (٦-١) مراحل اختبار النظام	٤٠
شكل (٧-١) طريقة الشلال	٤٤
شكل (٨-١) طريقة الشلال مع التداخل	٤٥
شكل (٩-١) أسلوب التكرار مع الزيادة	٤٧
شكل (١٠-١) النموذج الحلزوني لتطوير النظم	٤٨
شكل (١١-١) المنظور الديناميكي للعملية العقلانية الموحدة RUP	٤٩
شكل (١٢-١) تدفقات العمل الساكنة في العملية العقلانية الموحدة RUP	٥١
شكل (١٣-١) تطوير النظم باستخدام أسلوب التشييد المبني على النموذج	٥٣
الفصل الثاني:	
شكل (١-٢) التفكيك الوظيفي في مخططات DFDs	٥٨
شكل (٢-٢) الرموز المستخدمة في مخططات تدفق البيانات	٥٩
شكل (٣-٢) مخطط السياق لنظام الإعارة	٦٠
شكل (٤-٢) مخطط المنظور العام لنظام الإعارة	٦٢
شكل (٥-٢) التفكيك الوظيفي للعمليات	٦٣
شكل (٦-٢) مخطط تدفق البيانات التفصيلي لعملية "نفذ إعارة"	٦٥
شكل (٧-٢) مثال على استخدام أسلوب العربية الهيكلية	٦٧
شكل (٨-٢) الشكل الأولي لجداول القرارات لعملية "دقق عملية الإعارة"	٦٨
شكل (٩-٢) الشكل المبسط لجداول القرارات لعملية "دقق عملية الإعارة"	٦٩
شكل (١٠-٢) الشكل المبسط لشجرة القرارات لعملية "دقق عملية الإعارة"	٧٠
شكل (١١-٢) خصائص كائن "طالب"	٧٢
شكل (١٢-٢) تمثيل العلاقة بين الكائنات	٧٣

تابع - قائمة الأشكال

٧٤	شكل (١٣-٢) العلاقة الذاتية
٧٤	شكل (١٤-٢) العلاقة الثنائية
٧٦	شكل (١٥-٢) علاقة متعددة وكائن الربط
٧٧	شكل (١٦-٢) استخدام كائن الربط Associative Entity
٧٨	شكل (١٧-٢) استخدام أسلوب التعميم
٨٥	شكل (١٨-٢) الشكل العام للصنف Class
٨٦	شكل (١٩-٢) خصائص وعمليات الصنف order
٨٨	شكل (٢٠-٢) أنواع العلاقات في لغة UML
٨٨	شكل (٢١-٢) الارتباط بين أصناف الكائنات.
٨٩	شكل (٢٢-٢) صنف الارتباط
٨٩	شكل (٢٣-٢) الارتباط المؤهل
٩٠	شكل (٢٤-٢) مثال لخاصية التعميم
٩١	شكل (٢٥-٢) مثال للاعتمادية
٩٢	شكل (٢٦-٢) علاقة التجميع
٩٤	شكل (٢٧-٢) مثال لمخطط الأصناف
٩٦	شكل (٢٨-٢) مخطط حالات الاستخدام
٩٨	شكل (٢٩-٢) مثال لمحتوى حالة استخدام
٩٩	شكل (٣٠-٢) العلاقات بين حالات الاستخدام
١٠١	شكل (٣١-٢) العلاقات في مخطط حالات الاستخدام
١٠٣	شكل (٣٢-٢) مثال لمخطط التتابع
١٠٥	شكل (٣٣-٢) الانتقالات في مخطط الحالة
١٠٧	شكل (٣٤-٢) مخطط الحالة للكائن Ticket
١٠٨	شكل (٣٥-٢) مثال لاستخدام مخطط النشاط
١٠٩	شكل (٣٦-٢) تمثيل المكون في لغة UML2
١٠٩	شكل (٣٧-٢) تمثيل آخر للمكون في لغة UML2
١١٠	شكل (٣٨-٢) تمثيل علاقات التحقيق والتبعية باستخدام الأسهم
١١٠	شكل (٣٩-٢) تمثيل علاقات التحقيق والتبعية باستخدام المكبس والكرة
١١١	شكل (٤٠-٢) مثال لمخطط المكونات
١١٢	شكل (٤١-٢) تمثيل العقدة Node في لغة UML

تابع - قائمة الأشكال

١١٣	شكل (٤٢-٢) تمثيل عملية نشر المنتجات البرمجية
١١٣	شكل (٤٣-٢) دمج مخطط المكونات في مخطط النشر
١١٥	شكل (٤٤-٢) مخطط النشر لنظام إدارة الطلبة
١١٦	شكل (٤٥-٢) أيقونة الحزمة
١١٦	شكل (٤٦-٢) علاقة التعميم بين الحزم
١١٧	شكل (٤٧-٢) علاقة التبعية بين الحزم
١١٧	شكل (٤٨-٢) علاقة التحقق بين الحزم
١١٩	شكل (٤٩-٢) مخطط حزم الأصناف لجزء من نظام التسجيل الجامعي
الفصل الثالث:	
١٢٦	شكل (١-٣) تقديرات الوقت المتوقع لأنشطة أحد المشاريع
١٢٧	شكل (٢-٣) الترتيب الزمني لأنشطة أحد المشاريع
١٢٨	شكل (٣-٢) استخدام برنامج Microsoft Project لإعداد مخطط جانت
١٢٩	شكل (٤-٢) الشكل العام للعقدة في مخطط بيرت
١٣٠	شكل (٥-٢) حساب قيم الوقت المبكر والمتأخر والفائض لكل نشاط
١٣١	شكل (٦-٣) مخطط بيرت Pert لأحد المشاريع
١٣٢	شكل (٧-٢) استخدام برنامج Microsoft Project لإعداد مخطط بيرت
١٣٣	شكل (٨-٢) أداة ManagePro لإدارة الأداء
١٣٦	شكل (٩-٢) استخدام أداة Risk في تقدير المخاطر
١٣٨	شكل (١٠-٣) وثيقة المتطلبات
١٤٠	شكل (١١-٢) استخدام أداة النمذجة المرئية Rational Rose
١٤٠	شكل (١٢-٢) استخدام أدوات النمذجة لإنتاج التقارير
١٤١	شكل (١٣-٢) استخدام أداة Power Designer لنمذجة قواعد البيانات
١٤٢	شكل (١٤-٢) بيئة التطوير المتكاملة Sun ONE Studio
١٤٥	شكل (١٥-٢) استخدام أداة التطوير Oracle JDeveloper
١٤٨	شكل (١٦-٢) تطبيقات أدوات هندسة البرمجيات المدعومة بالحاسب
١٤٩	شكل (١٧-٢) مكونات أدوات هندسة البرمجيات المدعومة بالحاسب
الفصل الرابع:	
١٥٢	شكل (١-٤) عملية إدارة مشاريع البرمجيات
١٥٧	شكل (٢-٤) مخطط أنشطة تخطيط المشروع

تابع - قائمة الأشكال

١٥٨	شكل (٣-٤) قائمة مهام المشروع
١٦٠	شكل (٤-٤) جدول مهام أحد المشاريع باستخدام خريطة جانت
١٦١	شكل (٥-٤) تقويم الموارد
١٦٢	شكل (٦-٤) حساب تكلفة المشروع بناءً على جدول المشروع
١٦٦	شكل (٧-٤) قياس الإنتاجية في طريقة COCOMO II
١٦٨	شكل (٨-٤) الخواص المؤثرة في تحديد قيمة المجهود
١٦٩	شكل (٩-٤) استخدام مخطط جانت لتابعة جدول المشروع
١٧٠	شكل (١٠-٤) حساب التكلفة المبدئية والفعالية لمهام المشروع
١٧٢	شكل (١١-٤) تحليل القيمة المكتسبة
١٨٣	شكل (١٢-٤) خطة إزالة المخاطر
١٨٩	شكل (١٣-٤) مفهوم عملية إدارة التشكيل والتغيير
الفصل الخامس:	
٢٠٠	شكل (١-٥) مخطط السياق لنظام الأعمال
٢٠١	شكل (٢-٥) نموذج لمخطط حالات الاستخدام لنظام الأعمال
٢٠٢	شكل (٣-٥) نموذج لمخطط الأصناف لنظام الأعمال العام
٢٠٤	شكل (٤-٥) نموذج لمخطط حالات استخدام المجال
٢٠٥	شكل (٥-٥) نموذج لمخطط الأصناف للمجال
٢٠٧	شكل (٦-٥) نموذج حالات الاستخدام لنظام إدارة البريد الإلكتروني
٢٠٩	شكل (٧-٥) أحد قوالب وثيقة حالات الاستخدام
٢١٢	شكل (٨-٥) مخطط الأصناف للتطبيق EM
الفصل السادس:	
٢١٥	شكل (١-٦) التبعيات الدائرية بين الحزم
٢١٥	شكل (٢-٦) إزالة التبعيات الدائرية
٢١٦	شكل (٣-٦) تبعيات الطبقة
٢١٨	شكل (٤-٦) تبعيات الصنف
٢١٩	شكل (٥-٦) تبعيات التوريث في أثناء الترجمة
٢١٩	شكل (٦-٦) تبعيات التوريث في أثناء التنفيذ
٢٢٠	شكل (٧-٦) تبعيات الطرق
٢٢١	شكل (٨-٦) تبعية التطبيق

تابع - قائمة الأشكال

٢٢٢	شكل (٩-٦) مثال على التبعية الدائرية
٢٢٣	شكل (١٠-٦) استخدام صنف الواجهة لحل التبعية الدائرية
٢٢٥	شكل (١١-٦) إطار MVC
٢٢٦	شكل (١٢-٦) إطار PCMEF
٢٢٧	شكل (١٣-٦) نمط الواجهة
٢٢٨	شكل (١٤-٦) نمط المصنع المجرد
٢٢٩	شكل (١٥-٦) نمط سلسلة المسؤوليات
٢٣٠	شكل (١٦-٦) نمط المراقب
٢٣١	شكل (١٧-٦) نمط الوسيط
الفصل السابع:	
٢٣٤	شكل (١-٧) تصميم جدول الأفلام
٢٣٥	شكل (٢-٧) جملة SQL لإنشاء جدول الأفلام
٢٣٦	شكل (٣-٧) نموذج منطقي لقاعدة بيانات الأفلام
٢٣٧	شكل (٤-٧) النموذج المفاهيمي لقاعدة بيانات الأفلام
٢٣٩	شكل (٥-٧) قواعد العمل في النموذج المنطقي
٢٣٩	شكل (٦-٧) زناد الحذف على جدول الأفلام
٢٤٠	شكل (٧-٧) برمجة قواعد البيانات من خلال التطبيق
٢٤١	شكل (٨-٧) استخدام الإجراءات المخزنة Stored Procedures
٢٤٢	شكل (٩-٧) النموذج المفاهيمي لقواعد بيانات الجامعة
٢٤٣	شكل (١٠-٧) استخدام لغة ODL لتوصيف قاعدة بيانات الجامعة
٢٤٥	شكل (١١-٧) تحويل علاقة واحد - إلى - متعدد
٢٤٦	شكل (١٢-٧) تحويل العلاقة متعدد - إلى - متعدد
٢٤٧	شكل (١٣-٧) تحويل علاقة التعميم
٢٤٨	شكل (١٤-٧) مخطط الأصناف للتطبيق EM
٢٤٩	شكل (١٥-٧) النموذج المادي لقواعد بيانات التطبيق EM
٢٥٠	شكل (١٦-٧) ملف تعليمات SQL لإنشاء قواعد بيانات التطبيق EM
٢٥١	شكل (١٧-٧) محتوى عينة قاعدة البيانات EM
الفصل الثامن:	
٢٥٦	شكل (١-٨) جدول إيجاد الأصناف من حالات الاستخدام للتطبيق EM

تابع - قائمة الأشكال

٢٥٧	شكل (٢-٨) النموذج الأولي للأصناف للتطبيق EM
٢٥٩	شكل (٣-٨) توسعة جدول إيجاد الأصناف
٢٦٠	شكل (٤-٨) مخطط الأصناف للتطبيق EM
٢٦١	شكل (٥-٨) مخطط الكائنات للتطبيق EM
٢٦٣	شكل (٦-٨) أنواع الرسائل في مخطط التابع
٢٦٤	شكل (٧-٨) تبادل الرسائل في مخطط الاتصال
٢٦٥	شكل (٨-٨) المنظور العام لمخطط التفاعل
٢٦٧	شكل (٩-٨) مخطط التفاعل "login"
٢٦٨	شكل (١٠-٨) مخطط التفاعل "Exit"
٢٦٩	شكل (١١-٨) مخطط التفاعل "view Unsent Messages"
٢٧١	شكل (١٢-٨) مخطط التفاعل "Display Message Text"
٢٧٢	شكل (١٣-٨) مخطط التفاعل "Email Message"
٢٧٣	شكل (١٤-٨) مخطط التفاعل "Incorrect User Name or Password"
٢٧٤	شكل (١٥-٨) مخطط التفاعل "Incorrect Option"
٢٧٥	شكل (١٦-٨) مخطط التفاعل "Too Many Messages"
٢٧٦	شكل (١٧-٨) مخطط التفاعل "Email Could Not be Sent"
الفصل التاسع:	
٢٧٩	شكل (١-٩) تصميم الصنف Movie
٢٨٠	شكل (٢-٩) محتوى البرنامج Movie.java
٢٨٢	شكل (٣-٩) تحويل نموذج الأصناف إلى النموذج التطبيقي
٢٨٣	شكل (٤-٩) واجهة مجموعة Collection Interface
٢٨٤	شكل (٥-٩) تمثيل الصنف Actor بلغة جافا
٢٨٥	شكل (٦-٩) تمثيل الصنف ListedAs بلغة جافا
٢٨٦	شكل (٧-٩) الاتصال بقواعد البيانات في لغة جافا
٢٨٧	شكل (٨-٩) برنامج البحث بلغة جافا
٢٨٨	شكل (٩-٩) برنامج الاتصال وتنفيذ عملية البحث في قواعد البيانات
٢٨٩	شكل (١٠-٩) إطار عام لنموذج الاختبار
٢٩١	شكل (١١-٩) دور الاختبار في تطوير التطبيقات
٢٩٢	شكل (١٢-٩) وثيقة سيناريو اختبار القبول للتطبيق EM

تابع - قائمة الأشكال

الفصل العاشر:	
٣٠٢	شكل (١-١٠) استخدام النوافذ في واجهات المستخدم
٣٠٣	شكل (٢-١٠) استخدام القوائم في واجهة التطبيق
٣٠٤	شكل (٣-١٠) نموذج الأصناف لأدوات التحكم في أداة Swing
٣٠٥	شكل (٤-١٠) أدوات التحكم في أداة Swing
٣٠٦	شكل (٥-١٠) معالجة أحداث واجهة المستخدم
٣٠٩	شكل (٦-١٠) مثال لصفحة HTML
٣١٠	شكل (٧-١٠) محتويات ملف HTML
٣١٢	شكل (٨-١٠) مثال على استخدام لغة Java Script داخل ملف HTML
٣١٤	شكل (٩-١٠) نموذج حالة البرمجيات
٣١٤	شكل (١٠-١٠) استخدام البرمجيات
٣١٦	شكل (١١-١٠) الاتصال بين العميل وخادم الويب في تقنية Servlet
٣١٧	شكل (١٢-١٠) خطوات استدعاء وتنفيذ Servlet
٣١٧	شكل (١٣-١٠) مثال على استخدام ال Servlet
٣١٨	شكل (١٤-١٠) مثال لاستخدام تقنية JSP
٣١٩	شكل (١٥-١٠) شاشة الدخول للتطبيق EM
٣٢٠	شكل (١٦-١٠) الصفحة الرئيسية لبريد المستخدم
٣٢١	شكل (١٧-١٠) عرض محتويات الرسائل
٣٢٢	شكل (١٨-١٠) تعديل محتويات الرسالة
٣٢٢	شكل (١٩-١٠) إنشاء رسالة جديدة

مقدمة:

بسم الله والصلاة والسلام على رسول الله وعلى آله وصحبه وسلم، وبعد.

لقد شهدت العشرون عاماً الماضية طفرة كبيرة في معظم البلدان العربية في استخدام تقنية المعلومات في مجالات الحياة المختلفة مثل المجالات الاقتصادية والصناعية والزراعية والتعليمية والصحية والاجتماعية والخدمية والأمنية وحتى المجالات الإعلامية والترفيهية التي تحتوى على عدد كبير جداً من تطبيقات الأعمال التي تحتوى بدورها على مجموعة كبيرة من إجراءات العمل ويتم تطبيقها على كمية كبيرة من البيانات بواسطة مجموعة من المستخدمين من خلال نظم تشغيل مختلفة. يتطلب ذلك توافر العديد من الأساليب والأدوات والموارد التي تدعم عمليات تطوير البرمجيات المناسبة لتلك التطبيقات التي يجب أن تتميز بإمكانية الاتساع وسهولة الصيانة بحيث يمكننا إضافة متطلبات جديدة في أى وقت أو التعديل في المتطلبات الحالية.

ولأن عمليات تطوير البرمجيات للأنظمة كبيرة الحجم تستغرق وقتاً كبيراً ويشارك فيها عدد كبير من الأفراد وتستخدم العديد من الموارد المادية، فإنها تحتاج إلى إدارة جيدة للسيطرة على تلك العوامل المؤثرة في نجاح مشروع البرمجيات، حيث تهتم إدارة مشاريع البرمجيات بعمليات التخطيط والمراقبة والتحكم في التغيرات التي تطرأ على المشروع وكذلك السيطرة على الموارد المستخدمة. وتؤدي غالباً الإدارة الجيدة للمشروع إلى إنتاج البرمجيات المطلوبة في الوقت المحدد وبالجودة المطلوبة وفي حدود الميزانية المقدرة للمشروع، مما يتطلب استخدام مجموعة من الأساليب والأدوات الحديثة التي تدعم عمليات إدارة وتطوير وإنتاج مشاريع البرمجيات.

ولقد ظهرت في السنوات القليلة الماضية العديد من الأساليب والأدوات التي يمكن استخدامها في هذا المجال، وسوف نركز في هذا الكتاب على أسلوب تطوير البرمجيات المبني على الكائنات وهو أسلوب يلائم المشاريع كبيرة الحجم والقابلة للاتساع، وسوف نستعين ببعض التطبيقات والحالات الدراسية لتعزيز عملية فهم كيفية استخدام تلك الأساليب، كما سنعرض أيضاً بعض أساليب وأدوات إدارة مشاريع البرمجيات.

هذا، ويحتوى الكتاب على عشرة فصول في جزأين: يتناول الجزء الأول المفاهيم الأساسية لتطوير وإدارة مشاريع البرمجيات والأدوات المستخدمة وذلك من خلال الفصول الأربعة الأولى، حيث نقدم في الفصل الأول عرضاً لمراحل دورة حياة تطوير

البرمجيات مع وصف مختصر لطرق تطوير البرمجيات. ونعرض في الفصل الثاني أساليب نمذجة البرمجيات مع التركيز على أسلوب النمذجة المبني على الكائنات. ونعرض في الفصل الثالث لبعض أدوات هندسة البرمجيات مثل أدوات إدارة مشاريع البرمجيات وأدوات نمذجة البرمجيات وأدوات تطوير البرمجيات. ونقدم في الفصل الرابع ببعض التفصيل عمليات وإجراءات إدارة مشاريع البرمجيات. ويتناول الجزء الثاني من الكتاب عمليات تطوير مشاريع البرمجيات باستخدام الأسلوب المبني على الكائنات. حيث نعرض في الفصل الخامس لكيفية نمذجة متطلبات نظام الأعمال مع الاستعانة بحالة دراسية لتعزيز الفهم. ونقدم في الفصل السادس بعض الأطر المعمارية المستخدمة في تصميم النظام. ونتناول في الفصل السابع كيفية تصميم قواعد بيانات النظام. كما نقدم في الفصل الثامن طرق تصميم الأصناف والتفاعلات بينها داخل النظام. وفي الفصل التاسع نوضح كيفية استخدام لغة البرمجة «جافا» في تطوير البرمجيات بالإضافة إلى عرض لمراحل اختبار البرمجيات المنتجة. ونقدم في الفصل العاشر عرضاً لطرق تصميم واجهات استخدام النظام مع التركيز على استخدام تقنية الإنترنت لعرض البرمجيات المنتجة على الويب.

ونأمل أن يساعد هذا الكتاب طلاب العلم في الوطن العربي في مجال تقنية المعلومات في التعرف على كيفية استخدام الأساليب والأدوات الحديثة في تطوير وإدارة البرمجيات، وأن يكون إضافة جادة للمكتبة العربية في هذا المجال، وعلى الله قصد السبيل.

المؤلف: د. عبد الوهاب كامل السماك

sammaka@ipa.edu.sa

sammaka@gmail.com

الجزء الأول

مشاريع البرمجيات

Software Projects

تعتمد معظم بلدان العالم الآن على استخدام الأنظمة المبنية على الحاسب الآلى فى إنشاء وتشغيل وصيانة مشاريع البنية الأساسية الخاصة بها فى القطاعات المختلفة مثل قطاعات الصناعة والزراعة والصحة والتعليم والتجارة والبنوك والخدمات وغيرها. لذلك فإن عملية إنتاج وصيانة البرمجيات بشكل اقتصادى تعتبر ضرورة ملحة للاقتصاد الوطنى والدولى. تركز عمليات هندسة البرمجيات على تطوير وإنتاج برمجيات عالية الجودة بشكل اقتصادى. وقد نشأت فكرة هندسة البرمجيات فى أواخر الستينيات عند دراسة الأزمات والمشكلات التى بدأت تصاحب عملية تطوير البرمجيات فى ظل التقدم المستمر فى تقنيات الحاسب الآلى، وكذلك زيادة حجم التطبيقات المطلوبة واحتوائها على قدر كبير من التعقيدات. ولقد أوضحت تلك الدراسة أن السبب الرئيسى فى تلك الأزمات يرجع إلى ضعف الأساليب المنهجية التى يتم تطوير البرمجيات على أساسها، مما أدى إلى إنتاج البرمجيات فى وقت أكبر من المتوقع وبتكلفة أكبر بكثير من التكلفة المقدرة وبكفاءة تشغيل ضعيفة، بالإضافة إلى صعوبة صيانتها فيما بعد. وقد تطلب ذلك استحداث طرق وأساليب وأدوات جديدة تساهم فى تطوير وإنتاج البرمجيات كبيرة الحجم بكفاءة وجودة عالية وبشكل اقتصادى أيضاً.

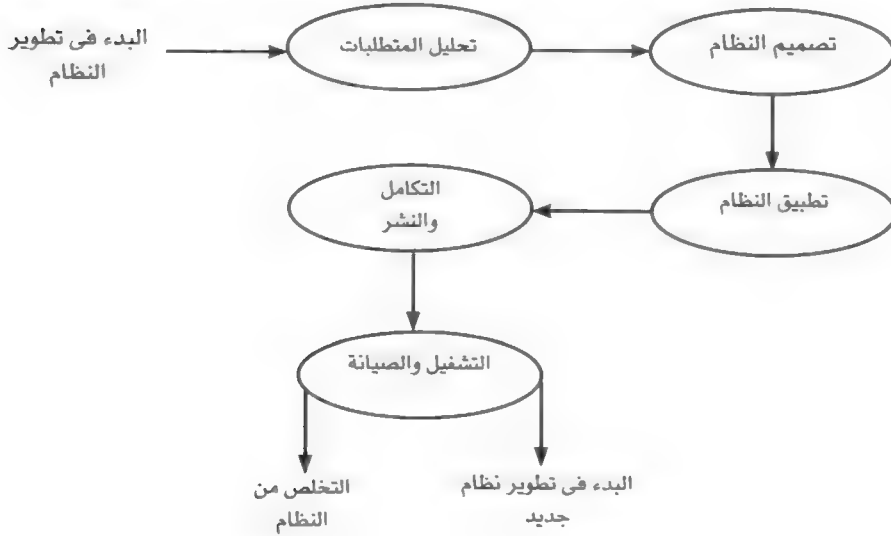
ولقد أدى ازدياد الاعتماد على استخدام الطرق الجديدة لهندسة البرمجيات فى بناء النظم، واستحداث تقنيات جديدة خصوصاً فى مجالات أنظمة الاتصالات واندماجها مع أنظمة الحاسبات، إلى زيادة المتطلبات والاحتياجات لأنظمة البرمجيات مما زادها تعقيداً. وعليه فقد أصبح الاستمرار فى عمليات التطوير والتحديث فى أساليب وأدوات هندسة البرمجيات أمراً ضرورياً لإمكانية السيطرة على عمليات تطوير وإنتاج البرمجيات المعقدة بشكل جيد. لأن مشاريع تطوير البرمجيات كبيرة الحجم تعتبر مكلفة وتستغرق وقتاً طويلاً ويقوم على تنفيذها فرق عمل مختلفة وتحتوى على قدر كبير من الخطورة، فإنها تحتاج إلى إدارة جيدة لى تمكثها من تحقيق أهدافها بشكل دقيق. وسوف نعرض فى هذا الجزء بعض المفاهيم الأساسية لتطوير وإدارة مشاريع البرمجيات والأدوات المستخدمة فيها.

الفصل الأول

دورة حياة تطوير البرمجيات

Software Development Lifecycle

كما أن لكل كائن حي دورة حياة تبدأ بولادته وتنتهى بوفاته، فإن للبرمجيات دورة حياة تصف التغيرات التى تحدث للمنتج البرمجى خلال مراحل تطويره المختلفة منذ نشأته واستخدامه وصيانه إلى أن تصبح عملية صيانه والتحديث فيه صعبة وتكون جدوى الاستمرار فى استخدامه ضعيفة ويتم الاستغناء عنه. ومن المعروف أن تلك التغيرات تحدث بالتدرج فى شكل مراحل تطوير تتسم بالنمو التدريجى والتحديث والتوسعة المستمرة، حتى يتم اتخاذ القرار بالاستغناء عنه واستبداله بنظام آخر. ومن الجدير بالذكر أن عملية الإحلال أو التحويل إلى نظام آخر بديل تتم أيضاً بالتدرج. يوضح شكل (١-١) المراحل المختلفة لدورة حياة البرمجيات، والتى سوف يتم استعراضها بالتفصيل لاحقاً فى هذا الفصل.



شكل (١-١) دورة حياة البرمجيات

تتكون دورة حياة البرمجيات من المراحل التالية (Maciaszek, 2005):

- ١- تحليل المتطلبات Requirements Analysis.
- ٢- تصميم النظام System Design.
- ٣- التطبيق Implementation.
- ٤- التكامل والنشر Integration and Deployment.
- ٥- التشغيل والصيانة Operation and Maintenance.

يتضح من الشكل (١-١) أنه في حالة التقديم لمنتج برمجى لمؤسسة ما وبمروره بمراحل التطوير المختلفة واستخدامه فعليا وازدياد اعتماد المؤسسة عليه فى إنجاز الأعمال وحتى وصوله إلى حالة عدم المقدرة على تحقيق المتطلبات الجديدة للمؤسسة أو أن تكون صيانة النظام كبيرة جدا ومكلفة إلا أنه من الصعب جداً أن تتخذ المؤسسة القرار بالتخلص منه والعودة إلى النظام اليدوى التقليدى لذلك يكون من الأفضل البدء فى تطوير نظام جديد، والتخطيط لاستبدال النظام الحالى بنظام آخر يحقق متطلبات المؤسسة من خلال تطبيق مراحل التطوير مرة أخرى من البداية، وهو ما يسمى بدورة حياة المنتج البرمجى. ومن الضروري أن تتم عملية الإحلال بالتدرج، أى أنه يتم الاستمرار فى استخدام النظام الحالى فى أثناء مرحلة تطوير النظام الجديد وتثبيته واستخدامه، ويتم استخدام النظامين، القديم والجديد، لفترة حتى يتم التأكد من جودة وفائدة النظام الجديد فى تحقيق متطلبات المؤسسة، وبعد ذلك يتم الاستغناء عن النظام القديم.

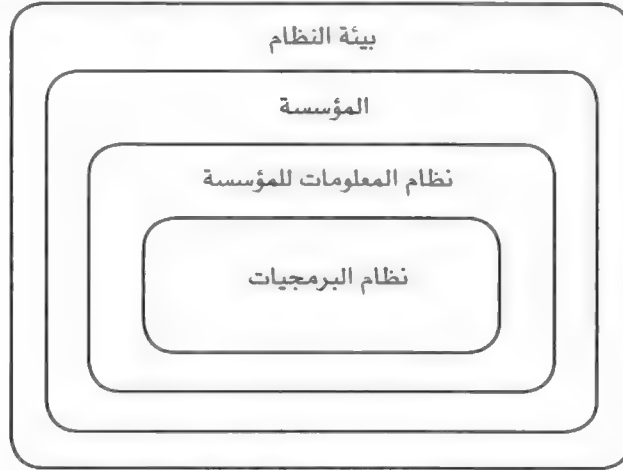
١-١ أساسيات هندسة البرمجيات Software Engineering Fundamentals

سوف نبدأ الآن بعرض أساسيات هندسة البرمجيات وذلك قبل المضى فى شرح تفاصيل دورة حياة البرمجيات. وسنقوم باستعراض تلك الأساسيات من خلال مجموعة من المفاهيم الأساسية التالية:

- النظام البرمجى Software System.
 - العملية البرمجية Software Process.
 - هندسة البرمجيات Software Engineering.
 - نمذجة البرمجيات Software Modeling.
- وسوف نستعرض الآن كل مفهوم بشئ من التفصيل.

١-١-١ النظام البرمجي Software System

يعتبر النظام البرمجي جزءاً من نظام أكبر وهو نظام المعلومات الخاص بالمؤسسة Enterprise Information System، وهو يعتبر أحد مكونات المؤسسة، وهي بدورها تعتبر جزءاً من بيئة نظم الأعمال، كما هو موضح بالشكل (٢-١). وهذا يعنى أن عملية تطوير النظام البرمجي تعتبر أحد أنشطة تطوير نظام المعلومات الخاص بالمؤسسة.



شكل (٢-١) مخطط بيئة نظم الأعمال

تهتم نظم المعلومات بعملية إنتاج وإدارة المعلومات وتقديمها للمستخدمين، مما يمكنه من إنجاز أعماله بكفاءة وجودة أعلى من ذي قبل، وتساعد على التطوير والانتفاع واتخاذ القرارات الصحيحة في الوقت المناسب مما يعود بالنفع على نظام الأعمال. يحتوى نظام المعلومات على مجموعة من المكونات المترابطة مثل:

- الأفراد.
- البيانات والمعلومات.
- الإجراءات وقواعد العمل.
- البرمجيات.
- الأجهزة.
- الاتصالات.

فمن الطبيعي أن يحتوى نظام المعلومات على مجموعة من القواعد وإجراءات العمل التى يتم تطبيقها على مجموعة من البيانات أو المعلومات، وذلك من خلال مجموعة المستخدمين الذين يقومون بتشغيل النظام فعلياً باستخدام أجهزة الحاسبات الآلية وملحقاتها وتقوم بتنفيذ مجموعة من البرمجيات، التى قد تحتاج إلى الاتصال بأجهزة أخرى تمدّها بالمعلومات فى أثناء التشغيل.

٢-١-١ العملية البرمجية Software Process

المقصود بالعملية Process عموماً هو الكيفية التى يتم بها إنجاز المهام أو أنشطة العمل المطلوبة لإنتاج منتج أو تقديم خدمة. ومن المعروف أيضاً أن البرمجيات Software تساهم بشكل فعال فى إنجاح نظام الأعمال Business وذلك من خلال الدور الذى تقوم به العملية البرمجية فى إصدار المنتج أو تقديم الخدمة المطلوبة بكفاءة. وعلى هذا فإن العملية البرمجية تعتبر جزءاً أساسياً من نظام الأعمال، حيث يتم تقديم هذه الخدمة خلال المستويات الإدارية المختلفة مثل مستوى التشغيل Operational Management، والمستوى التكتيكي Tactical Management، والمستوى الإستراتيجى Strategic Management، كما هو موضح فى الشكل (٣-١).



شكل (٣-١) العملية البرمجية للمستويات الإدارية المختلفة

تقوم العملية البرمجية غالباً بتقديم الخدمة لأحد المستويات الإدارية المذكورة. فعلى مستوى التشغيل تقوم العملية البرمجية بمعالجة البيانات الأولية وإنجاز المهام الأساسية لنظام الأعمال مثل معالجة طلبات العملاء وإصدار الفواتير وغيرها، وهو الجزء الأكبر من نظم الأعمال ويتم الاستعانة فيه بتقنية قواعد البيانات Data Bases.

وعلى المستوى التكتيكي يتم تحليل المعلومات المستخلصة من مستوى التشغيل والاستفادة منها في توجيه نظام الأعمال مثل تحليل ودراسة طلبات العملاء في فترة معينة، وكذلك اتجاه المبيعات في فترة ما واتخاذ القرارات التصحيحية المناسبة لتحسين عائدات نظام الأعمال، ويتم ذلك بالاستعانة بتقنية مخازن البيانات Data Warehouse. وعلى المستوى الإستراتيجي يتم تقنين المعرفة المستخلصة من المستويين السابقين من خلال مجموعة من الحقائق والقواعد التي تصف نظم المعرفة Knowledge Systems لنظام الأعمال، ويتم ذلك بالاستعانة بتقنية قواعد المعرفة Knowledge Bases. وبهذا يتضح الدور الذي تقوم به العملية البرمجية خلال المستويات الإدارية الثلاثة والقيمة التي تقدمها لنظام الأعمال.

٣-١-١ هندسة البرمجيات Software Engineering

تعتبر هندسة البرمجيات هي أحد أشكال النظم الهندسية التي تهتم بدراسة المبادئ والأسس التي تتحكم في تطوير وتشغيل وصيانة النظم المعقدة مثل النظم الميكانيكية والنظم الكهربائية والنظم الإنشائية وغيرها. ولكن تختلف هندسة البرمجيات عن باقي النظم الهندسية في عدة أشياء أساسية نذكر منها ما يلي:

المنتج Product

يكون المنتج غالباً في النظم الهندسية على شكل مواد Materials سواء كانت صغيرة الحجم مثل الأدوات التي نستخدمها في حياتنا اليومية والأجهزة الدقيقة، أو كانت كبيرة الحجم مثل المباني والكبارى ومحطات الكهرباء والسدود وغيرها. أما المنتج في هندسة البرمجيات فيكون في شكل برمجيات فقط سواء كانت صغيرة الحجم وقليلة الاستخدام مثل البرمجيات الخاصة بالأفراد التي تساعد في إنجاز أعمالهم بشكل ما، أو كانت كبيرة الحجم معقدة تستخدم على نطاق واسع لعدد من المستخدمين في أنحاء العالم مثل برامج نظم التشغيل و مترجمات لغات البرمجة ونظم قواعد البيانات وغيرها.

المبادئ Principles

يتم صياغة المبادئ المكتشفة في هندسة النظم في شكل نماذج رياضية أو نظريات مثبتة يتم تطبيقها في أثناء عملية تطوير المنتجات الهندسية. أما المبادئ المكتشفة

فى هندسة البرمجيات فيتم صياغتها فى عدة أشكال، منها ما يأخذ الشكل الرياضى المنطقى ومنها ما يأخذ شكل المنطق المضبب (غير الواضح) Fuzzy Logic، ومنها ما يأخذ شكل التوجيه لاكتشاف الأشياء Heuristic الذى يتحقق بدون قواعد مثبتة معتمداً على خبرات سابقة.

الإنتاج Production،

بينما يتم إنتاج المنتجات الهندسية من خلال عمليات التصنيع، فإن إنتاج البرمجيات يتم من خلال عمليات التطوير Development. هناك نوعان أساسيان للمنتج البرمجى هما:

برمجيات عامة: هى برمجيات تقوم بتنفيذ الوظائف الأساسية لتشغيل واستخدام الحاسب ويتم استخدامها على نطاق واسع للعديد من المستخدمين مثل نظم التشغيل Operating Systems ومعالجة النصوص Word Processing ونظم إدارة قواعد البيانات Database Management Systems (DBMS) وغيرها. ومن المعروف أنه يتم تطوير هذه الأنظمة من خلال شركات عالمية فى مجال صناعة البرمجيات وهى تقوم بإنتاج آلاف من النسخ المتطابقة للمنتج الواحد بغرض التوزيع.

برمجيات مخصصة: هى نظم برمجيات يتم تطويرها لتلائم متطلبات عميل معين وتعمل فى بيئة تشغيل معينة، مثل نظم الأعمال التجارية والإدارية والتحكم الخاصة بالمؤسسات أو المنشآت الخاصة ذات الطابع التجارى أو الإدارى أو التكنى.

ويتفق الهدف من المنتج البرمجى مع الهدف من المنتج الهندسى فى كونهما يحققان متطلبات وأهداف العميل والمستخدم بكفاءة وجودة معينة وأنهما يتسمان بإمكانية الصيانة. ولكنهما يختلفان فى إمكانية التوسعة. فبينما تتسم البرمجيات بقدرتها على الاتساع Extension والترقية Upgrade حسب التغيرات التى تنشأ فى بيئة العمل، فإن هذه العملية غير ممكنة فى أغلب النظم الهندسية حيث يتم تطوير نظام آخر جديد ومطور بدلاً من النظام المستخدم حالياً.

وعلى هذا فإنه يمكننا تعريف هندسة البرمجيات بأنها أحد مجالات علوم الحاسب الآلى التى تهتم بعملية إنتاج نظم البرمجيات كبيرة الحجم التى تحتوى على قدر من التعقيد مما يتطلب اشتراك فريق عمل أو فرق عمل متخصصة فى أثناء تطويرها. يتكون فريق العمل القائم على تطوير النظام البرمجى من مجموعة أفراد يمتلكون

مجموعة من الخبرات والمهارات التي تمكنهم من إنجاز المهام المطلوبة منهم بكفاءة وجودة عالية، بالإضافة إلى التفاعل الإيجابي والتكامل فيما بينهم في أثناء عمليات التطوير المختلفة. بالطبع لا تقتصر المهارة المطلوبة لأفراد فريق العمل على مهارة البرمجة فقط ولكن هناك مهارات أساسية أخرى مطلوبة لبناء النظام مثل مهارات تحليل نظام الأعمال وتحديد المتطلبات وتصميم الحلول وتصميم قواعد البيانات وتصميم واجهات المستخدم واختبار الوحدات المنتجة واختبار النظام العام ونشر وتوزيع النظام، بالإضافة إلى مهارات تحديد مواصفات الأجهزة المطلوبة وتصميم الشبكات ومهارات استخدام أدوات تطوير وإدارة المشروع. ولهذا فإن القائمين على تطوير أنظمة هندسة البرمجيات يجب أن يتوافر لديهم العديد من المهارات التي تؤهلهم للمشاركة الفعالة في واحد أو أكثر من أنشطة تطوير وإدارة مشاريع البرمجيات. ولذلك فإن مجال هندسة البرمجيات يعتبر من المجالات الغنية بالمعرفة والمهارات والأدوات والتقنيات التي تستخدم بشكل جيد من قبل فرق العمل لإنتاج برمجيات تحقق متطلبات نظم الأعمال في المجالات المختلفة.

٤-١-١ نمذجة البرمجيات Software Modeling

تعتمد هندسة البرمجيات على أسلوب النمذجة Modeling باعتباره أسلوباً فعالاً لوصف الأنشطة التي تتم خلال مراحل التطوير المختلفة لنظام البرمجيات، حيث تستخدم النماذج في تمثيل أو وصف أحد المكونات الوظيفية الفعلية لنظام الأعمال أو جزء منه وذلك بشكل مرئي بسيط، ومن خلال منظور معين يركز على أحد المفاهيم أو الأساليب المستخدمة لتطوير النظم. كما تستخدم النماذج أيضاً بوصفها وسيلة فعالة للتفاعل والتواصل بين أفراد فريق العمل بالمشروع وكذلك مع المستفيد. يستخدم مهندس البرمجيات أسلوب النماذج التوضيحية خلال المراحل المختلفة لدورة حياة تطوير النظم البرمجية مثل:

نموذج المتطلبات Requirements Model

هو نموذج يصف متطلبات المستفيد الوظيفية باستخدام إحدى طرق النمذجة المنهجية. كمثال على ذلك نستخدم مخطط تدفق البيانات DFD في الأسلوب الهيكلي، ونستخدم مخططات حالات الاستخدام Use Cases والأصناف Classes والتتابع Sequence والتفاعل Interaction وغيرها في الأسلوب المبني على الكائنات (الفصل الثاني).

النموذج المعماري Architectural Model،

هو نموذج يصف معمارية النظام البرمجي من خلال تمثيل النظم الفرعية التي يتكون منها النظام والتفاعلات فيما بينها مثل مخططات الحزم Packages والمكونات Components من خلال مجموعة من الأطر المعمارية المختلفة (الفصل السادس).

نموذج التصميم التفصيلي Detailed Design Model،

هو نموذج يصف المكونات التفصيلية لكل من البرمجيات Software والأجهزة Hardware المطلوبة لتطبيق الحلول المقترحة للنظام البرمجي مثل تصميم الأصناف والتفاعلات (الفصل الثامن) وتصميم مخطط النشر (الفصل الثاني).

ومن الممكن أن تتم تجزئة أى نموذج من تلك النماذج إلى عدة نماذج أخرى تفصيلية مثل نماذج وصف منطق العمليات ونماذج قواعد البيانات ونماذج واجهات المستخدم وغيرها.

تعتمد نوعية النماذج المستخدمة على المنهجية المستخدمة فى تطوير النظام البرمجي. وهناك مفهومان أساسيان يمكن استخدامهما لتطوير النظم هما:

المفهوم الوظيفي Functional Paradigm،

يقوم هذا المفهوم على أساس تجزئة النظام إلى وحدات فرعية صغيرة حتى يتم السيطرة عليها وتطويرها بسهولة، وبعد ذلك يتم تجميع الوحدات الفرعية لتكوين النظام العام. ويستخدم هذا المفهوم نماذج تدفق البيانات Data Flow Models لوصف عمليات النظام.

مفهوم الكائنات الموجهة Object-Oriented Paradigm،

يقوم هذا المفهوم على أساس تجزئة النظام إلى مكونات وظيفية صغيرة تسمى الكائنات Objects مع توصيف لطرق الاتصال بينها كما هو الحال فى الواقع الفعلى لنظام الأعمال. يستخدم هذا المفهوم نمذجة الأصناف Classes Modeling ونمذجة حالات الاستخدام Use-Case Modeling وغيرها من النماذج. وسوف يتم عرض هذه النماذج بالتفصيل فى الفصل القادم.

٢-١ مراحل دورة حياة تطوير البرمجيات Software Development Lifecycle Phases

تعتبر دورة حياة البرمجيات وصفاً موجزاً لإستراتيجية تطوير البرمجيات. فهي تصف مراحل وخطوات وأنشطة وطرق تطوير مشروع البرمجيات، بالإضافة إلى تمثيل المخرجات المتوقعة للمشروع خلال المراحل المختلفة. تتضمن دورة حياة تطوير البرمجيات المراحل التالية:

- تحليل المتطلبات Requirement Analysis.

- تصميم النظام System Design.

- التطبيق Implementation.

- التكامل والنشر Integration and Deployment.

- التشغيل والصيانة Operation and Maintenance.

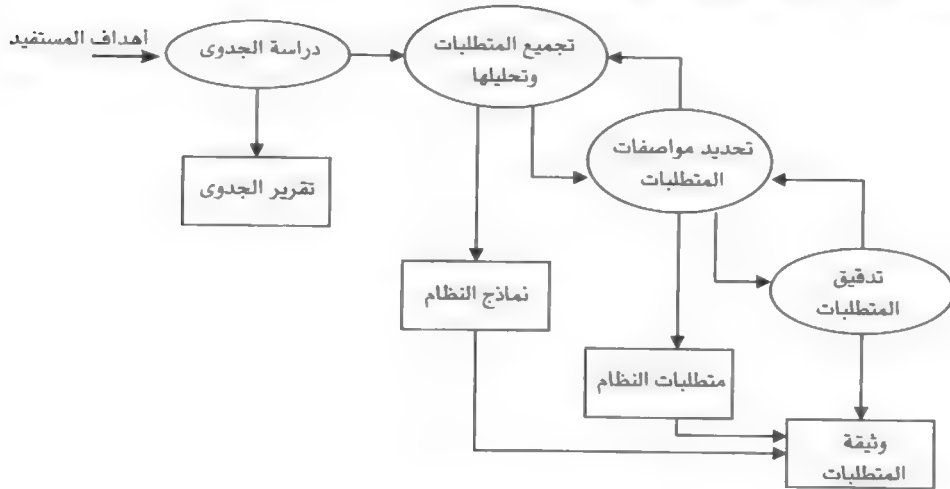
تتضمن كل مرحلة من هذه المراحل العديد من الأنشطة، ولكل مرحلة مدخلاتها ومخرجاتها وتأثيرها في جودة المنتج النهائي. وسوف نستعرض الآن كل مرحلة بشيء من التفصيل.

١-٢-١ تحليل المتطلبات Requirements Analysis

تعتبر عملية دراسة وتحليل متطلبات مستخدمى النظام هى أول خطوة فى عملية تطوير نظم البرمجيات. تحتوى هذه العملية على مجموعة من الأنشطة التى تهدف إلى تحديد متطلبات واحتياجات مستخدمى النظام. المقصود بمتطلبات المستخدمين هو مجموعة الخدمات التى يجب أن يقدمها النظام للمستخدمين، مع وصف لمجموعة القيود التى سوف تحكم عملية تنفيذ الخدمات. ولذلك تعتبر عملية تحديد المتطلبات من التحديات المهمة التى تواجه عملية تطوير البرمجيات وخصوصاً فى الحالات التى يكون فيها المستخدمون غير متفهمين لاحتياجاتهم بدقة ووضوح. فأحياناً يبالغ المستخدم فى وصف المتطلبات أو يحدد احتياجات تتعارض مع احتياجات أخرى لمستخدم آخر داخل النظام. وأحياناً أخرى لا يعطى المستخدم هذه العملية الأهمية المطلوبة فتكون النتيجة إنتاج برمجيات لا تقى بالخدمات المطلوبة وتحتاج إلى تعديل وإضافة الكثير مما يتطلب وقتاً ومجهوداً وميزانية إضافية عن المقدرة بالمشروع. من ثم فإن القائمين على تطوير النظام يعانون بشدة من مسألة عدم فهم مستخدمى النظام

للمتطلبات بدقة ووضوح مما يتطلب تدخل الأفراد القائمين على تحليل المتطلبات واستخدام خبراتهم ومهاراتهم المكتسبة من أنظمة أخرى مشابهة لمساعدة المستخدمين في تحديد متطلباتهم. تؤدي عملية تحديد المتطلبات إلى تعريف شكل النظام المطلوب وتحديد الوظائف والمهام التي يجب أن يقوم بها النظام بكفاءة. هناك مجموعة من الطرق والأساليب التي يتم اتباعها في أثناء عملية تحديد متطلبات النظام مثل:

- المقابلات مع المستخدمين وخبراء في المجال.
 - تصميم استبانات للمستخدمين.
 - ملاحظة المستخدمين في أثناء أدائهم للمهام.
 - دراسة وثائق ومستندات النظام الحالي.
 - دراسة أنظمة مشابهة للتعلم منها واكتساب معارف خاصة بالمجال.
 - إعداد نماذج تجريبية لتجربة الحلول المقترحة والتأكد من جدواها.
 - عقد جلسات عمل بمشاركة فريق التطوير وباقي المشاركين في المشروع.
- تتضمن عملية تحديد متطلبات النظام أربعة أنشطة رئيسية، كما هو موضح بالشكل (٤-١)، وهي (Sommerville, 2004):



شكل (٤-١) عملية تحديد المتطلبات

١- دراسة الجدوى Feasibility Study

تهدف دراسة الجدوى إلى تحديد مدى إمكانية تحقيق أهداف الاستفادة باستخدام التقنيات المتاحة وفي حدود الميزانية المقدرة للمشروع. تنتهي الدراسة بتوصيات تساعد المستفيد على اتخاذ القرار بالمضي قدماً في تنفيذ المشروع أو التوقف لعدم جدواه.

٢- تجميع المتطلبات وتحليلها Requirements Collection & Analysis

يتم في هذا النشاط تجميع معلومات عن متطلبات النظام من خلال ملاحظة النظام الحالي ومناقشة المستخدمين ودراسة النماذج والوثائق الخاصة بالنظام. وبعد ذلك يتم إعداد النماذج التي تصف بعض متطلبات النظام الوظيفية والهيكلية، والتي تساعد في عملية فهم النظام ومداولته مع باقي المشاركين في المشروع بشكل أفضل.

٣- تحديد مواصفات المتطلبات Determining Requirements Specification

يتم في هذا النشاط تحويل المعلومات المجمعة في النشاط السابق إلى وثيقة تصف جميع متطلبات النظام. هناك نوعان من المتطلبات هما:

- متطلبات وظيفية Functional Requirements: عبارة عن مجموعة الوظائف والعمليات التي يجب أن يقدمها نظام الأعمال في شكل مجموعة من الخدمات المقدمة للمستفيد من النظام.

- متطلبات غير وظيفية Nonfunctional Requirements: عبارة عن مجموعة من القيود التي تحكم تنفيذ المتطلبات الوظيفية لنظام الأعمال مثل بيئة التشغيل وقيود الأمن والسلامة وغيرها.

٤- تدقيق المتطلبات Requirements Validation

يتم في هذا النشاط التحقق من واقعية Realism واتساق Consistency واكتمال Completeness المتطلبات.

يتم استخدام أسلوب النمذجة Modeling لتوصيف المتطلبات التي تم تحديدها، وذلك باستخدام إحدى طرق النمذجة القياسية مثل لغة النمذجة الموحدة UML التي يتم من خلالها وصف متطلبات النظام في شكل مجموعة من المخططات Diagrams، بالإضافة إلى مجموعة من النماذج النصية Textual Models. ومن المعروف أن النظم المعقدة لا يمكن فهمها بشكل جيد ومتكامل باستخدام وجهة نظر أو من منظور

واحد Viewpoint، ولكن يتم تحليل المتطلبات من خلال وجهات نظر مختلفة تساعد على تعزيز عملية فهم وتوصيف متطلبات النظام بشكل جيد. وفي حالة استخدام إحدى أدوات هندسة البرمجيات المدعومة بالحاسب CASE فإنه يمكننا تخزين جميع النماذج التخطيطية والنصية التي تصف متطلبات النظام في المستودع الرئيسى للنظام Repository الذى يتم الاستفادة منه فى جميع مراحل تطوير النظام، بالإضافة إلى إنتاج وثائق النظام System Documentations بشكل آلى ودقيق. وتجدر الإشارة هنا إلى أنه فى حالة عدم استخدام تلك الأدوات المساعدة، فإنه يجب على مدير المشروع أن يركز على متابعة فرق التطوير بالمشروع، وضرورة استكمال توثيق العمليات التى يقومون بتنفيذها، ومن أولها وأهمها تحديد متطلبات النظام. يمكننا أيضاً الاستعانة ببعض القوالب Templates الخاصة بتوثيق المتطلبات المقدمة من الشركات العالمية المتخصصة أو تصميم قالب توثيق خاص بالمؤسسة. يحدد القالب غالباً الهيكل العام للوثيقة، بالإضافة إلى مجموعة من الإرشادات عن كيفية تعبئتها. تحتوى الوثيقة على النماذج والنصوص التى تصف مجموعة الخدمات التى يجب أن يقدمها النظام، بالإضافة إلى وصف لمجموعة القيود التى يتضمنها النظام فى أثناء تأدية تلك الخدمات مثل الأمن والأداء وظروف التشغيل وبعض القيود القانونية. وللتحقق من جودة مخرجات عملية تحديد المتطلبات، يتم مراجعة واختبار تلك المخرجات من قبل مجموعة متخصصة فى المجال تابعة لإدارة ضمان جودة البرمجيات بالمؤسسة Software Quality Assurance. تتكون هذه المجموعة من خبراء متخصصين فى اختبار البرمجيات وهم من خارج فريق التطوير للمشروع، وهم بذلك يصبحون مشاركين فى مسؤولية ضمان جودة البرمجيات المنتجة.

٢-٢-١ تصميم النظام System Design

تهتم عملية تصميم البرمجيات بالأنشطة الخاصة بوصف كل من هيكل البرمجيات، هيكل البيانات، وواجهات استخدام النظام. أحياناً لا يتم وصف خوارزميات العمليات بشكل مفصل فى أثناء مرحلة التصميم وذلك لإتاحة قدر أكبر من الحرية للمبرمجين فى أثناء مرحلة التطبيق. وفى حين كانت عملية التحليل غير مقيدة بأى قيود تخص بيئة التشغيل الأساسية التى سوف يعمل النظام من خلالها سواء كانت تخص العتاد Hardware أو تخص البرمجيات Software، فإن عملية التصميم تأخذ فى الاعتبار عدة اعتبارات وقيود خاصة ببيئة التشغيل التى سيعمل النظام من خلالها. فى الواقع العملى لا يوجد خط فاصل واضح بين مرحلتى التحليل والتصميم وذلك لسببين أساسيين. السبب الأول هو أن الطرق الحديثة لتطوير البرمجيات تعتمد على أسلوب

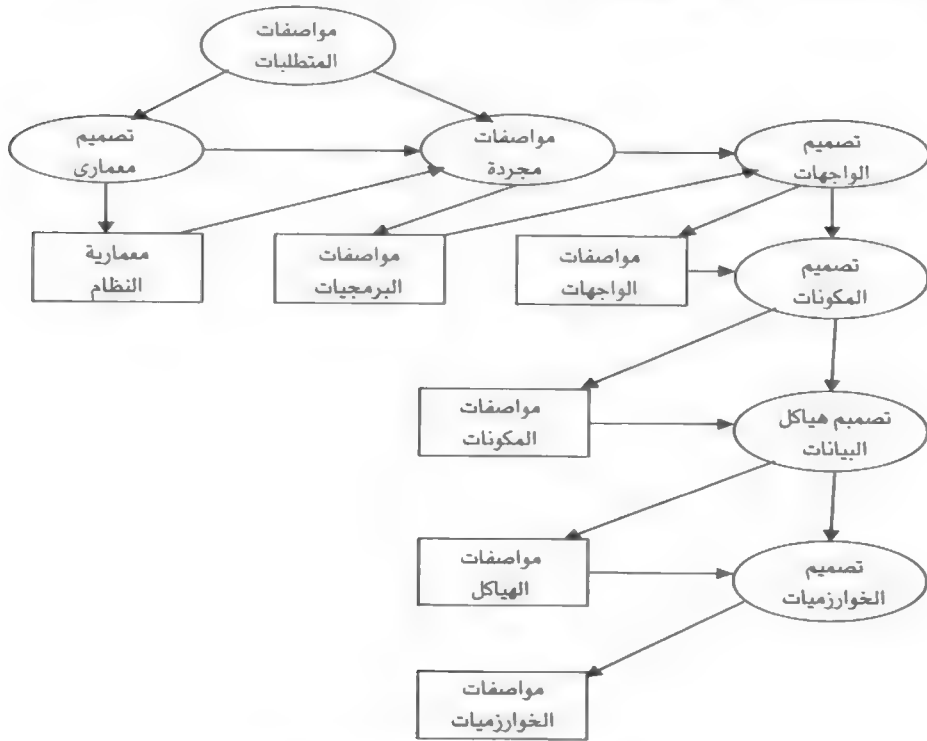
التكرار مع الزيادة Iterative with Increment وهو يتسم بإمكانية ظهور عدة إصدارات مختلفة للنظام تحت التطوير، ويكون أحد الإصدارات في مرحلة التحليل وإصدار آخر في مرحلة التصميم وإصدار ثالث في مرحلة التطبيق، وهكذا. والسبب الثاني يرجع لطبيعة أسلوب النمذجة المستخدم، فمثلاً يتسم أسلوب لغة النمذجة الموحدة UML بأنه يتم استخدام بعض النماذج لكل من عمليتي التحليل والتصميم. وعلى هذا فإنه يمكننا القول إن الانتقال من عملية التحليل إلى عملية التصميم يتم بأسلوب الإسهاب Elaboration وذلك بإضافة مواصفات تفصيلية إلى النموذج المستخدم في إنشاء عملية التحليل وليس بأسلوب التحويل من نموذج إلى نموذج آخر. يسمى هذا النوع باسم «التصميم التفصيلي Detailed Design». هناك نوع آخر من التصميم يسمى «التصميم المعماري Architectural Design» وهو يعنى بإعداد الإطار المعماري للنظام وهيكل المكونات للتصميم التفصيلي وأطر الاتصالات الداخلية بين المكونات. ونستطيع أن نوجز القول بأن الهدف الأساسي للتصميم المعماري هو الوصول إلى نظام مفهوم قابل للصيانة والاتساع. تتكون عملية التصميم من مجموعة الأنشطة التالية، كما هو موضح بشكل (١-٥) (Sommerville, 2004).

- تصميم معماري Architectural Design.
- مواصفات مجردة Abstract Specifications.
- تصميم الواجهات Interface Design.
- تصميم المكونات Component Design.
- تصميم هياكل البيانات Data Structure Design.
- تصميم الخوارزميات Algorithm Design.

١-٢-٣ التطبيق Implementation:

على الرغم من أن المقصود بتطبيق النظام هو برمجته، إلا أن عملية برمجة النظام لا تعنى مكوث مجموعة من المبرمجين في مكان واحد والقيام بكتابة البرامج باستخدام إحدى لغات البرمجة. فكما ذكرنا من قبل أن عملية التصميم لا تقوم بتوصيف جميع الخوارزميات التفصيلية، ولكن يتم تركها لمرحلة البرمجة، حيث يقوم المبرمج بتنفيذ العمليات حسب رؤيته والأسلوب البرمجي الذي يفضلُه ويحسن استخدامه، مع الأخذ في الاعتبار أن يتم ذلك طبقاً للمواصفات المذكورة في وثائق التحليل والتصميم. ومن

النادر اليوم أن يقوم المبرمج بكتابة أكواد جميع البرامج، ولكنه يستعين بمكتبة البرامج واستخدام بعض المكونات البرمجية الجاهزة، مما يوفر الوقت والمجهود. وهذا يتطلب أن يكون المبرمج على دراية جيدة بمحتويات مكتبة البرامج وكيفية استخدامها بشكل جيد. يقوم المبرمج أيضاً بعمليات الهندسة الأمامية والهندسة العكسية في أثناء عملية التطبيق وخصوصاً إذا كان مستخدماً لإحدى أدوات هندسة البرمجيات المدعومة بالحاسب (CASE) Computer Aided Software Engineering وفى الحقيقة، إن هذه الأدوات تساعد كثيراً فى عمليات إنتاج قواعد البيانات وتوليد البرامج الأساسية طبقاً لمواصفات التصميم التفصيلية التى تم إدخالها، وبعد ذلك يقوم المبرمج بالتدخل بإضافة الأكواد الضرورية لربط المكونات البرمجية المنتجة وتعديلها حسب الحاجة.



شكل (٥-١) نموذج عام لعمليات التصميم

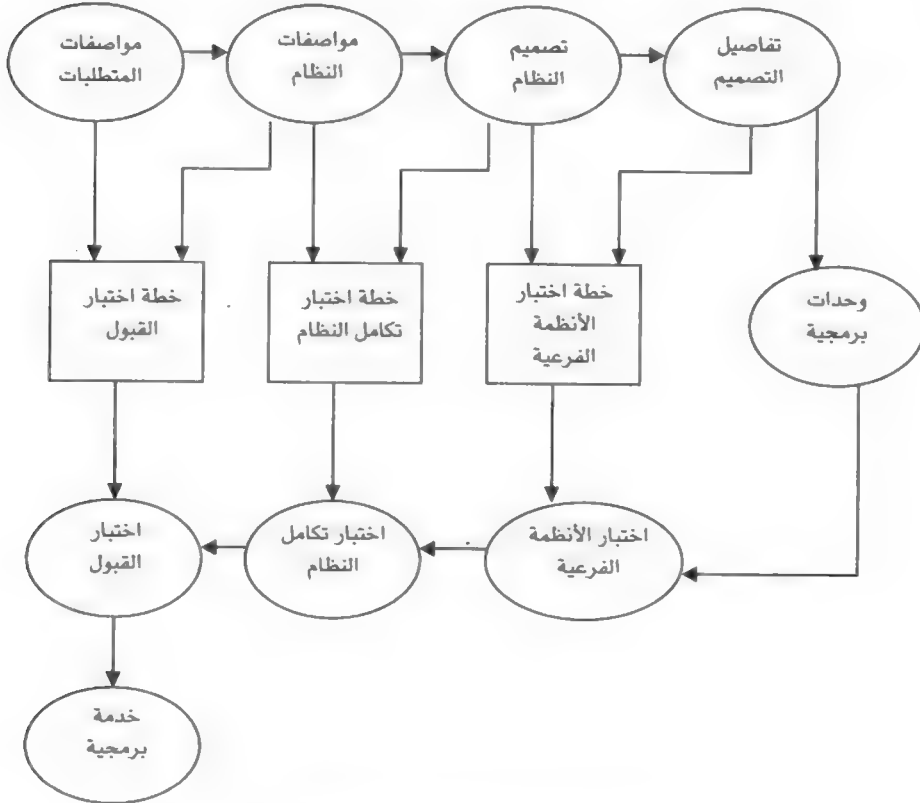
وبعد الانتهاء من عملية اختبار المنتجات البرمجية يقوم المبرمج بعملية إعادة هندسة البرمجيات المنتجة عكسياً لتعديل مواصفات التصميم المدخلة من قبل، وذلك لضمان

التوافق بين مواصفات التصميم والبرمجيات المنتجة، وقد يتطلب ذلك مشاركة كل من المبرمجين والمصممين في مراجعة هذه العملية وإجراء التعديلات الضرورية على كل من الأكواد ومواصفات التصميم. من المعروف أن مرحلة التطبيق تستغرق الوقت الأكبر خلال مراحل تطوير النظام، والسبب في ذلك لا يرجع إلى حجم الأكواد المكتوبة فحسب إنما يرجع أيضاً إلى كثرة الأخطاء والإضافات والتعديلات التي تتم على الأكواد في أثناء عمليات المراجعة والاختبار للبرمجيات المنتجة. أحياناً يقوم المبرمج بنفسه بعمليات المراجعة والاختبار لمنتجاته، ولكن من الأفضل أن يقوم بها فريق آخر، إذ يتضمن مجموعة من محللي النظام والمستخدمين ويقومون بفحص واختبار البرامج المنتجة باستخدام بيانات حقيقية ومعدة مسبقاً لاختبار البرامج ومقارنة النتائج بما هو متوقع.

١-٢-٤ التكامل والنشر Integration and Deployment

تتضمن عملية التكامل أنشطة تجميع الأنظمة الفرعية Sub-Systems التي تم تطويرها واختبارها من قبل. من الممكن أن يتم تجميع جميع الأنظمة الفرعية معاً في آن واحد لتكوين النظام العام، ولكن من الأفضل أن تتم عملية التكامل بالتدريج وبشكل متزايد مما يضمن نجاحها في وقت أقل وبمجهود أقل من الطريقة الأولى. فمن عيوب الطريقة الأولى أنها تتطلب أن يتم الانتهاء من تطوير واختبار جميع الأنشطة الفرعية أولاً، وهو طلب غير عملي وخاصة بالنسبة للأنظمة الكبيرة. بالإضافة إلى أن عملية تكامل أو تجميع عدد كبير من الأنظمة الفرعية في آن واحد تحتاج إلى مجهود كبير لاكتشاف الأخطاء التي تحدث في أثناء عملية التكامل، وتستغرق عملية التغلب عليها مجهوداً ووقتاً كبيراً. في حين أن إجراء عملية التكامل بالتدريج يوفر المجهود والوقت؛ لأن الأخطاء التي تحدث في أثناء تكامل نظام فرعي معين مع نظام حالي تكون غالباً من النظام الجديد أو نتيجة التفاعلات بين النظامين ويمكن اكتشافها والتغلب عليها بسهولة. وبعد نجاح المرحلة الأولى لعملية التكامل يتم إضافة نظام فرعي آخر للنظام الحالي وتكرار هذه العملية حتى يتم الحصول على النظام العام. ولضمان إتمام عملية التكامل للنظام بنجاح، يتم اختبار التفاعلات والاتصالات بين الأنظمة الفرعية، ومن ثم اختبار أداء النظام العام وهو ما يسمى باختبار التكامل Integration Test. يوضح شكل (١-٦) مراحل اختبار النظام في أثناء عمليات التطوير والتسليم. والجدير بالذكر أن طريقة تنفيذ عملية التكامل

تتطابق بدرجة كبيرة مع التصميم المعماري للنظام، إذ إن هذا التصميم يوضح مكونات النظام والاعتمادية بينها. ومن الممكن أن تتم عملية التكامل بين مكونات النظام من أعلى إلى أسفل Top-Down أو من أسفل إلى أعلى Bottom-Up أو بأسلوب وسط وهو خليط بين الأسلوبين يسمى Middle-Out.



شكل (٦-١) مراحل اختبار النظام

تهتم عملية النشر Deployment بأنشطة تسليم النظام المطور للعميل وتثبيته في بيئة العمل الطبيعية للبدء في التشغيل الفعلي. تتم هذه العملية بالتدرج كما هو الحال في عملية التكامل، فيتم نشر البرمجيات المنتجة على مراحل في شكل إصدارات مختلفة. فيتم تثبيت الإصدار الأول للنظام واختباره من قبل المستخدمين في بيئة العمل الطبيعية، ويطلق على هذا النشاط اسم اختبار ألفا Alpha Test. يتم تحديث هذا الإصدار عند ظهور أي أخطاء أو ملاحظات من المستخدمين تستدعي

التعديل فيه. وبعد ذلك يتم إضافة مكونات أخرى للنظام إلى الإصدار الحالي ويتم اختبارها أيضاً في بيئة العمل الطبيعية وبذلك يتم إنتاج إصدار ثانٍ للنظام. يتم تكرار تلك العملية حتى نصل إلى إنتاج الإصدار الأخير للنظام وهو يحتوى على جميع مكونات النظام العام. وفى النهاية يتم اختبار النظام العام من قبل المستخدمين للتأكد من تحقيقه لجميع الأهداف والمتطلبات المذكورة فى مواصفات الاحتياجات للنظام، وهو ما يسمى باختبار القبول Acceptance Test. تتضمن عملية نشر النظام عدة أنشطة أخرى مثل تدريب مستخدمى النظام وهى يمكن أن تبدأ قبل عملية نشر النظام فى الحالات التى تتطلب تدريب المستخدمين على استخدام أدوات أو تقنيات أخرى ضرورية لتشغيل النظام نفسه. ومن أنشطة عملية النشر أيضاً عملية إنتاج دليل استخدام النظام للمستخدمين (User Documentation (Guide).

٥-٢-١ التشغيل والصيانة Operation and Maintenance:

يبدأ المنتج البرمجى المنفذ الذى تم نشره عند العميل بتنفيذ إجراءات العمل اليومية أو ما يسمى بعمليات التشغيل اليومية وذلك فى ظل وجود النظام القديم. يتم تشغيل النظامين (القديم والجديد) على التوازي لبعض الوقت. وبعد التأكد من كفاءة تشغيل النظام الجديد فى إنجاز الأعمال اليومية يتم خروج النظام القديم من التشغيل واستمرار تشغيل النظام الجديد. ومع بداية التشغيل الفعلى للنظام تبدأ فعلياً عمليات الصيانة Maintenance للنظام. هناك ثلاثة أحوال يتم فيها صيانة النظام تحت التشغيل، هى:

- إصلاح العيوب والأخطاء التى يتم اكتشافها فى أثناء عمليات التشغيل.
- إجراء تعديلات فى البرمجيات المنتجة لتلائم التغييرات التى تحدث فى بيئة نظام الأعمال.
- إضافة إمكانيات جديدة للبرمجيات المنتجة لتحسين جودتها والارتقاء بها لصالح نظام الأعمال أو نتيجة لاتساع النظام نفسه.

من المعروف أن المؤسسات الكبيرة ترصد ميزانية سنوية لأعمال الصيانة لنظام الأعمال تساهم فى تحديث وتطوير النظام بشكل مستمر مما يحمى نظام الأعمال من التقادم أو الوصول إلى مرحلة التوقيف نتيجة انعدام الفائدة من استخدامه.

٣-١ طرق تطوير البرمجيات Software Development Methods:

كما ذكرنا من قبل فإن العملية البرمجية تحتوى على مجموعة من الأنشطة المترابطة التي تؤدي إلى إنتاج منتج برمجى معين، وهى عملية معقدة تختلف تفاصيلها من مؤسسة لأخرى ومن مشروع لمشروع آخر، إذ إنها تتأثر بعدة عوامل منها:

- خبرات ومهارات الأشخاص القائمين على تنفيذ الأنشطة.
- مجال التطبيق Application Domain لنظام الأعمال.
- التغيرات الخارجية التي تؤثر فى بيئة نظام الأعمال مثل التغيرات السياسية أو الاقتصادية أو الاجتماعية أو التقنية.
- التغيرات الداخلية لنظام الأعمال مثل التغيير فى الإدارة أو ظروف العمل أو الموقف المالى للمؤسسة أو فى أفراد فريق العمل.
- حجم مشروع نظام الأعمال.

ومع ذلك فإن معظم مشاريع نظم الأعمال تشترك فى احتوائها على مجموعة من الأنشطة الأساسية التى تستخدم فى تطوير نظم الأعمال وتؤدي إلى تسليم منتجات برمجية تفى بمتطلبات المستفيد من النظام. تتضمن الأنشطة الأساسية عمليات تحديد وتحليل مواصفات الاحتياجات، وتصميم الحلول المقترحة للنظام واختيار الأنسب منها، وتطبيق الحلول التى تم اختيارها، واختبار البرمجيات المنتجة وتكاملها، ثم عمليات نشر البرمجيات المنتجة والتحول إلى النظام الجديد، ثم صيانة النظام بعد بدأ التشغيل الفعلى.

إلا أن طريقة تطوير مشاريع البرمجيات تختلف حسب طبيعة المشروع وظروفه والقيود التى تحكمه. هناك العديد من الطرق والمنهجيات التى تستخدم لتطوير مشاريع البرمجيات. تمثل كل طريقة العملية البرمجية من منظور معين ويقدم معلومات حول العملية البرمجية. نعرض هنا لبعض الأساليب الشائعة الاستخدام فى تطوير مشاريع البرمجيات مثل (Maciaszek, 2005):

- أسلوب الشلال Waterfall.
- أسلوب التكرار مع الزيادة Iterative with Increment.
- هندسة البرمجيات المبنية على المكونات Component-Based Software Engineering.

- التشييد المبني على النموذج (Model Driven Architecture (MDA).

- أسلوب النماذج التجريبية Prototyping.

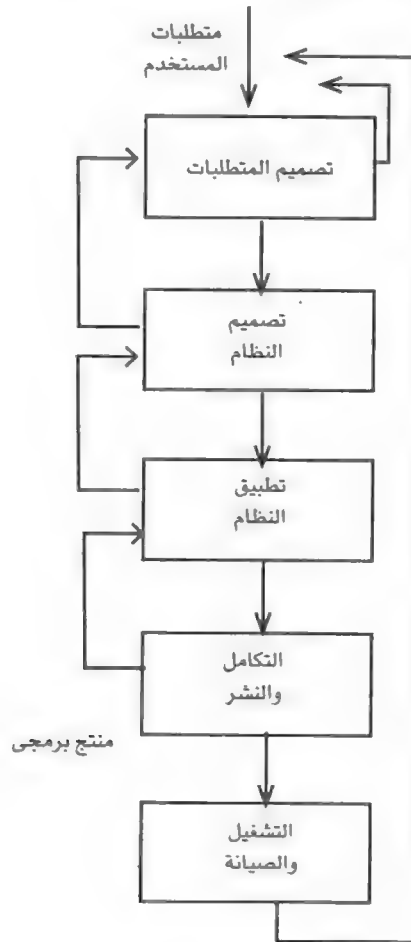
والجدير بالذكر هنا أنه يمكننا الاستفادة من استخدام أكثر من أسلوب في تطوير المشروع بخاصة المشاريع الضخمة. فعلى سبيل المثال هناك أسلوب تطوير جديد يسمى بالعملية العقلانية الموحدة (RUP) Rational Unified Process وهو يجمع عناصر عديدة من أغلب الأساليب السابقة الذكر. والآن سوف نقوم باستعراض كل أسلوب بشيء من التفصيل.

١-٣-١ طريقة الشلال Waterfall Method:

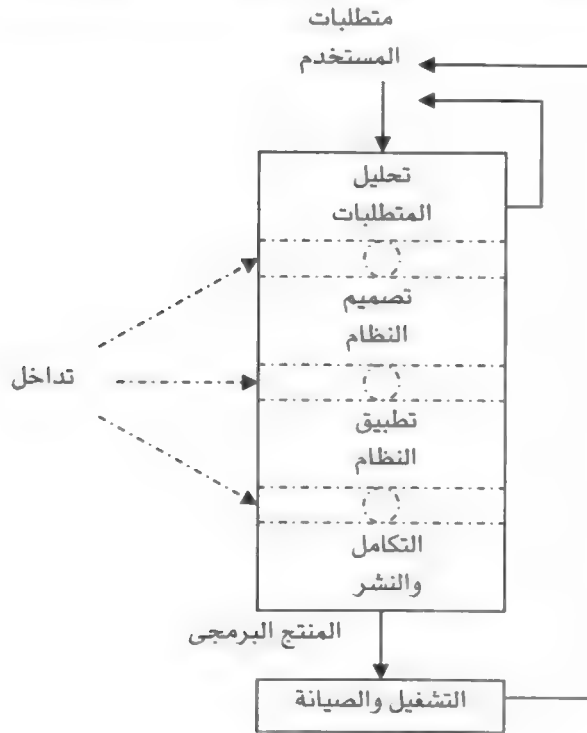
تعتبر طريقة الشلال من الطرق التقليدية التي شاع استخدامها لتطوير النظم فى السبعينيات وقد حققت نجاحاً كبيراً فى ذلك الوقت، حيث كانت تناسب حجم وطبيعة النظم وقتها. أما الآن فقد قل استخدامها نظراً للتطورات التي حدثت فى التقنيات المستخدمة ومن ثم فى المتطلبات. يوضح شكل (١-٧) المراحل المختلفة لطريقة الشلال مع إمكانية الارتداد (التغذية العكسية) Feedback بين المراحل. تتسم طريقة الشلال بأسلوب المراحل المتتالية التي لا يتم فيها الانتقال من مرحلة حالية إلى مرحلة تالية إلا بعد الانتهاء من تنفيذ جميع أنشطة المرحلة الحالية، مع إمكانية العودة من أى مرحلة إلى المرحلة السابقة لها لإجراء بعض التعديلات، ويمكن أن يستمر هذا الارتداد حتى الوصول إلى المرحلة الأولى. هناك عدة أشكال يمكن أن تأخذها طريقة الشلال، فمنها ما يسمح بالتداخل Overlap بين المراحل المتتالية كما هو موضح بشكل (١-٨)، وفيها يمكن لأى مرحلة أن تبدأ قبل اكتمال أنشطة المرحلة السابقة لها.

تتميز طريقة الشلال بأنها تقوم بإنتاج التوثيق الخاص بكل مرحلة فى نهاية المرحلة، مما يضمن اكتمال عملية توثيق النظام. وفى المقابل فإن طريقة الشلال تعاني من عدة مشاكل فى أثناء تطوير النظم الكبيرة التي تتسم بكثرة التغيرات فيها وتحتوى على قدر من التعقيد. وتكمن غالباً المشكلات من أسلوب التقسيم غير المرن الذي تستخدمه طريقة الشلال فى تجزئة عمليات تطوير النظام إلى عدة مراحل متعاقبة، مما يستلزم ضرورة اكتمال ووضوح جميع متطلبات العميل من النظام فى المرحلة الأولى وصعوبة تغييرها فيما بعد. فإذا استحدثت متطلبات جديدة فى مراحل متأخرة من المشروع، فإن ذلك سوف يتطلب البدء فى دورة حياة تطوير جديدة للنظام. وإذا حدث تعديل

فى المتطلبات فى أثناء عمليات التطوير، فإنه يمكن إجراؤها عن طريق الارتداد للمراحل السابقة، ولكن ذلك يؤدى إلى تعطل فرق العمل فى المراحل التالية نظراً لاعتمادها على المراحل السابقة لها. وعلى هذا يمكننا القول إن طريقة الشلال تناسب النظم الصغيرة أو النظم التى تتسم بوضوح واستقرار فى الأهداف والمتطلبات وضمان اكتمالها فى بداية المشروع.



شكل (٧-١) طريقة الشلال



شكل (٨-١) طريقة الشلال مع التداخل

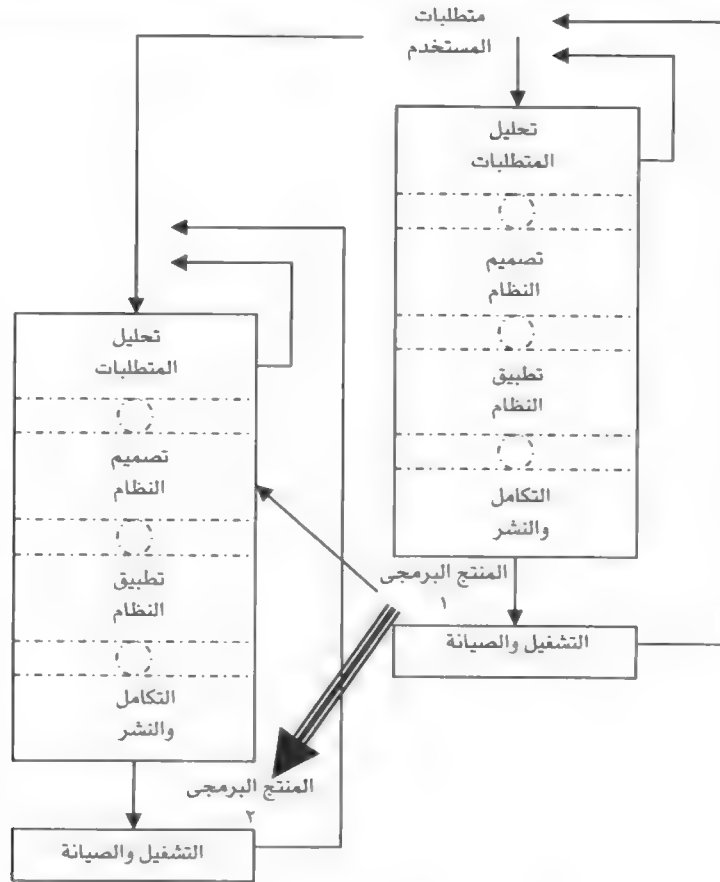
٢-٣-١ طريقة التكرار مع الزيادة Iterative with Increment

يقصد بالأسلوب التكرارى فى تطوير البرمجيات أنه يتم تكرار تنفيذ مجموعة من العمليات البرمجية مع إضافة مكونات جديدة فى كل مرة وذلك بهدف إثراء أو تنمية المنتج البرمجى. يلائم هذا الأسلوب تماماً النظم الكبيرة والمعقدة التى تتعرض لكثير من التغيرات فى بيئة العمل أو التى لا تكون متطلباتها كاملة فى بداية عملية التطوير. ففى تلك الأنظمة يمكننا بالاتفاق مع العميل تحديد مجموعة الخدمات أو الوظائف المطلوبة وتحديد أسبقية تنفيذ أولوية كل خدمة أو وظيفة حسب أهميتها بالنسبة للعميل. بعد ذلك يتم تحديد مواصفات الاحتياجات للخدمات ذات الأولوية القصوى أولاً والبداية فى تطويرها وتطبيق جميع عمليات التطوير عليها حتى يتم الحصول على النسخة الأولية للمنتج البرمجى، ويتم استخدامها. بعد ذلك نقوم بإضافة خدمة

جديدة أو أكثر ونكرر الإجراءات نفسها بدءاً من تحديد مواصفات الاحتياجات وباقي عمليات التطوير وانتهاء بالحصول على النسخة المطورة الثانية من المنتج البرمجي. وهكذا حتى ننتهي من إنجاز جميع الخدمات المطلوبة من النظام. وهنا نلاحظ أنه قد تم إنتاج النظام العام بالتدريج في شكل نسخ مطورة على مراحل تكرارية وليس من مرة واحدة كما هو الحال في طريقة الشلال. ونلاحظ أيضاً أنه قد تم تكرار تنفيذ مراحل التطوير كاملة لإنتاج كل نسخة مطورة. وفي حالات أخرى يتعرض نظام الأعمال لكثير من التغيرات سواء في الوظائف أو في ظروف العمل أو في التقنية المستخدمة، فإنها تحتاج إلى التعديل في مواصفات الاحتياجات ومن ثم تعديل في التصميم والتطبيق. ومن ثم إصدار نسخة مطورة للنسخة الحالية للمنتج البرمجي. وعلى هذا يتم تعريف أسلوب التطوير بالتكرار مع الزيادة بأنه الأسلوب الذي يؤدي إلى إصدار نسخ مطورة، سواء بالتحسين أو بالإضافة، للمنتج البرمجي وذلك في نهاية كل محاولة. يوضح شكل (٩-١) أسلوب التطوير بالتكرار مع الزيادة الذي يتضمن مراحل طريقة الشلال في كل محاولة تكرارية. ويتضح من الشكل أنه بعد الانتهاء من عملية التشغيل وفي أثناء عملية الصيانة تبدأ عملية التكرار بإضافة احتياجات جديدة والاستمرار في تنفيذ باقي المراحل في دورة حياة التطوير، وهكذا.

يوفر استخدام أسلوب التطوير بالتكرار مع الزيادة عدة مميزات مهمة مثل:

- عدم احتياج العميل للانتظار طويلاً حتى يتم الانتهاء من تطوير النظام العام مرة واحدة، ولكنه يستطيع استخدام النسخة الأولية في تنفيذ الاحتياجات الضرورية للنظام مبكراً.
- اكتساب العميل خبرة جيدة في أثناء استخدام النسخة الأولية للنظام مما يمكنه من تحديد باقي الاحتياجات والمتطلبات بشكل دقيق في النسخ المطورة التالية.
- تكرار عملية اختبار الخدمات والوظائف الأساسية للنظام في كل مرة يتم فيها اختبار النسخة الجديدة للنظام مما يزيد من جودة المنتج ومن ثم جودة النظام العام.
- يؤدي ذلك كله إلى التخفيف من المخاطر التي يمكن أن يواجهها النظام وتؤثر فيه ويمكن أن تؤدي إلى فشله أو انهياره. ويتم هذا التخفيف نتيجة للتغلب التدريجي على المشكلات التي تنشأ في كل محاولة وعدم تركها لتتراكم وتتفاقم في تأثيرها في النظام العام.



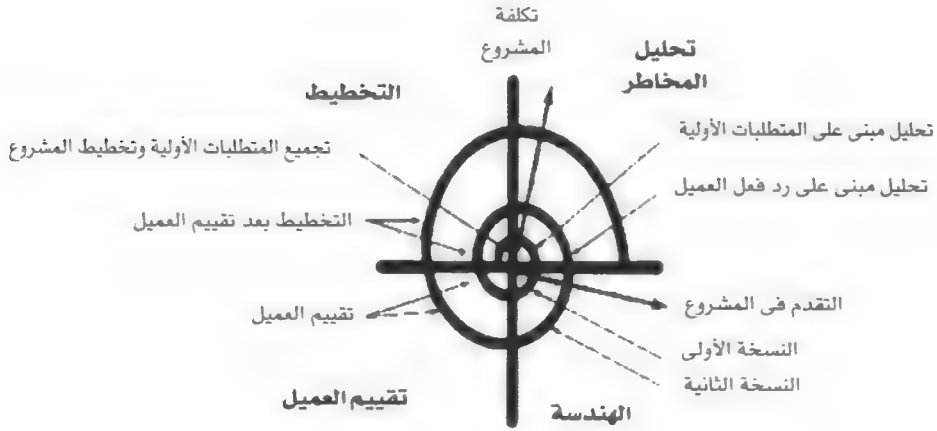
شكل (١-٩) أسلوب التكرار مع الزيادة

هناك طريقتان تنفذان هذا الأسلوب في تطوير النظم، الأولى تسمى «النموذج الحلزوني Spiral Model» وقد ظهرت في أواخر الثمانينيات، والثانية تسمى «العملية العقلانية الموحدة RUP» والمقدمة من شركة IBM في عام ٢٠٠٣.

١-٢-٣-١ النموذج الحلزوني Spiral Model،

يتم تمثيل العملية البرمجية في هذا النموذج بشكل حلزوني Spiral بدلاً من تمثيلها في شكل مجموعة من الأنشطة المتتالية. كل حلقة Loop في هذا النموذج تمثل مرحلة في العملية البرمجية مثل تحديد المتطلبات أو التصميم وغيرها. ينقسم

النموذج الحلزوني إلى أربعة قطاعات Sectors، كما هو مبين في الشكل (١٠-١)، وهى قطاع التخطيط Planning، قطاع تحليل المخاطر Risk Analysis، والقطاع الهندسى Engineering، وقطاع تقييم العميل Customer Evaluation. تبدأ دائماً الحلقة الأولى للنموذج من قطاع التخطيط وهى تمثل عملية تجميع المتطلبات الأولية ومن ثم تخطيط المشروع. بعد ذلك يدخل المشروع فى مرحلة (قطاع) تحليل المخاطر التى تبدأ بتحليل التكلفة/المنفعة وتحليل الفرص المتاحة للمساعدة فى عملية اتخاذ القرار بالاستمرار والانتقال إلى مرحلة التطوير (القطاع الهندسى) أو التوقف عن تنفيذ المشروع لعدم جدواه. يهتم القطاع الهندسى بعمليات التطوير Development التى تنتج النسخة الأولى للنظام البرمجى التى تخضع بعد ذلك لتقييم العميل (قطاع تقييم العميل). وبعد ذلك تبدأ الحلقة الثانية فى النموذج الحلزوني لإنتاج النسخة الجديدة المطورة للنظام.



شكل (١٠-١) النموذج الحلزوني لتطوير النظم

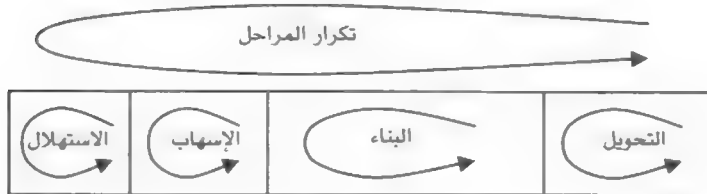
نلاحظ أن هذا النموذج قد تعامل مع عمليات التطوير من خلال قطاع واحد وهو القطاع الهندسى. ونلاحظ أيضاً التركيز على عملية تحليل المخاطر وتكرارها مع كل حلقة مما يساعد على الاكتشاف المبكر لأى مخاطر يمكن أن تواجه المشروع ويمكنه التغلب عليها بطريقة أسرع وأسهل من تراكمها ومن ثم يخفف من آثارها فى

المشروع. ونلاحظ أيضاً تكرار عملية التخطيط في بداية كل حلقة، وعملية التقييم للمنتج البرمجي من قبل العميل في نهاية كل حلقة، مما يحسن من جودة المنتج البرمجي. والملاحظة الأخيرة هي أن كل جزء من الحلقة داخل القطاع الهندسي تمثل محاولة تكرارية Iteration للمنتج البرمجي وهي تمثل مرحلة من مراحل التطوير حتى الوصول إلى المنتج النهائي.

١-٢-٣-٢ العملية العقلانية الموحدة (RUP) Rational Unified Process

تعتبر العملية العقلانية الموحدة أحد نماذج العملية الحديثة لتطوير النظم والتي تم اقتباسها من أسلوب لغة النمذجة الموحدة UML. وهي تعتبر مثلاً جيداً للعملية المدمجة Hybrid Process التي تحتوي على مجموعة من العناصر المدمجة من الطرق السابقة. فهي تحتوي على عدة مراحل مثل طريقة الشلال Waterfall، ومع ذلك فهي تدعم أسلوب التكرار مع الزيادة Iterative with Increment، بالإضافة إلى كونها تتسم بواقعية أكثر في أثناء عمليات تحديد المواصفات والتصميم للنظام. يمكننا وصف العملية العقلانية الموحدة من خلال ثلاثة منظورات هي:

- ١- منظور ديناميكي Dynamic يصف مراحل تطوير النظام حسب الوقت.
 - ٢- منظور ساكن Static يصف أنشطة العملية.
 - ٣- منظور عملي يقترح مجموعة من الممارسات الجيدة المكتسبة من خبرات سابقة لتطبيقها خلال عمليات التطوير للنظام الحالي.
- يصف المنظور الديناميكي مراحل تطوير النظام حسب الوقت وليس حسب الأنشطة، وهو يتكون من أربع مراحل أساسية كما هو مبين في الشكل (١-١١):



شكل (١-١١) المنظور الديناميكي للعملية العقلانية الموحدة RUP

١- مرحلة الاستهلال Inception Phase:

الهدف من هذه المرحلة هو تأسيس حالة نظام الأعمال وذلك من خلال تحديد جميع الكائنات الخارجية External Entities التى تتفاعل مع النظام، مع تعريف لتلك التفاعلات. يتم استخدام تلك المعلومات لتحديد الدور الذى يقوم به النظام تحت التطوير ضمن مشروع نظام الأعمال العام.

٢- مرحلة الإسهاب Elaboration Phase:

الهدف من هذه المرحلة هو فهم مجال التطبيق وتأسيس الإطار العام للنظام وإعداد خطة المشروع وتحديد النقاط التى تحتوى على بعض المخاطر بالمشروع. ويتوقع فى نهاية تلك المرحلة أن يتم إنجاز نموذج مواصفات المتطلبات للنظام والوصف المعماري للنظام، بالإضافة إلى إعداد خطة تطوير المشروع.

٣- مرحلة البناء Construction Phase:

تهتم تلك المرحلة بالأنشطة الخاصة بتصميم وبرمجة واختبار النظام. يتم إنجاز أجزاء من تلك الأنشطة على التوازي أو التداخل، ثم يتم التكامل بينها فى أثناء تلك المرحلة. وفى نهاية المرحلة يمكننا الحصول على نظام برمجى جاهز للتشغيل ومرفقاً معه نماذج توثيق النظام المطور.

٤- مرحلة التحول Transition Phase:

الهدف من تلك المرحلة هو الانتقال من مرحلة التطوير إلى مرحلة التشغيل فى بيئة التشغيل الطبيعية. وفى نهاية هذه المرحلة يكون لدينا نظام برمجى موثق يعمل بصورة صحيحة فى بيئة التشغيل الفعلية.

تدعم العملية العقلانية الموحدة RUP أسلوب التكرار مع الزيادة على مستويين هما:

داخلى: يتم التطوير بالتدرج باستخدام أسلوب الزيادة داخل كل مرحلة.

خارجى: يتم استخدام أسلوب الحلقة بين جميع المراحل المتتالية فى العملية العقلانية.

يركز المنظور الساكن للعملية العقلانية على الأنشطة التى يتم تنفيذها فى أثناء عملية التطوير. ويتم توصيف أنشطة المشروع من خلال مجموعة من تدفقات العمل

Workflows، وهى عبارة عن ستة تدفقات عمل أساسية وثلاثة تدفقات عمل مساندة، كما هو مبين فى الشكل (١-١٢). والجدير بالذكر هنا أن العملية العقلانية الموحدة RUP قد تم تقديمها بالاشتراك مع لغة النمذجة الموحدة UML، ولهذا فإننا سوف نجد أن وصف تدفقات العمل يكون موجهاً نحو نماذج UML المصاحبة لها.

تدفق العمل	الوصف
نمذجة نظام الأعمال	توصيف عمليات نظام الأعمال باستخدام نموذج حالات الاستخدام.
تحليل متطلبات النظام	توصيف متطلبات مستخدمى النظام من خلال تفاعل مجموعة الفاعلين مع حالات استخدام النظام.
التصميم	إنشاء نماذج التصميم المعمارى والمكونات وتتابع الكائنات للنظام.
التطبيق	تطبيق مكونات وإجراءات نظام الأعمال من خلال مجموعة من الوحدات البرمجية المتفاعلة.
الاختبار	اختبار كل وحدة برمجية منتجة وكذلك اختبار تكامل الوحدات داخل النظام.
النشر	إصدار المنتج البرمجى وتوزيعه وتثبيته عند العميل.
إدارة التشكيل والتغيير	دعم المنتج البرمجى وإدارة عمليات التغيير التى تطرأ عليه.
إدارة المشروع	إدارة عمليات تطوير مشروع نظام الأعمال.
بيئة المشروع	توفير مجموعة الأدوات المساعدة التى يحتاج إليها فريق العمل بالمشروع.

شكل (١-١٢) تدفقات العمل الساكنة فى العملية العقلانية الموحدة RUP

ويصف المنظور العملى للعملية العقلانية الموحدة مجموعة جيدة من الممارسات العملية لهندسة البرمجيات التى ينصح بتطبيقها فى تطوير النظم. نذكر هنا بعض الممارسات المهمة التى ينصح باستخدامها.

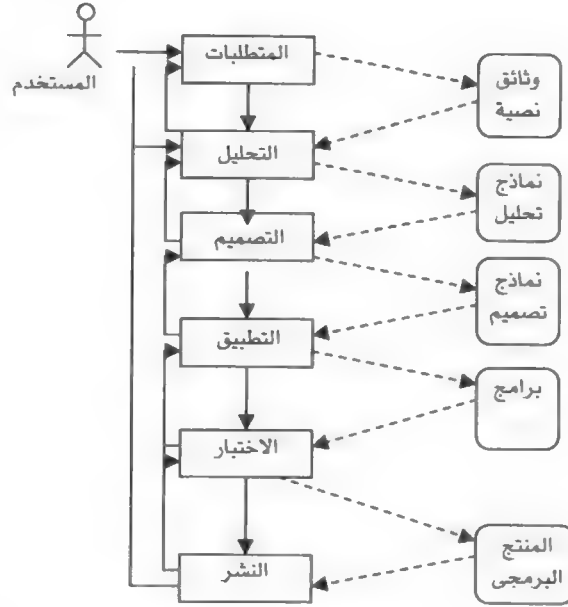
- يجب تطوير النظام البرمجى بالتدرج والزيادة، وذلك بناء على الأولويات التى تم تحديدها من قبل العميل، حيث نبدأ فى تطوير واختبار وتسليم المهام ذات الأولوية القصوى، ثم نضيف إليها مهام أخرى تليها فى الأولوية، وهكذا.
- يجب توثيق جميع متطلبات العميل بشكل دقيق، ومتابعة التغييرات التى تحدث فيها، وتحليل تأثير تلك التغييرات فى النظام قبل قبول تنفيذها.

- من الأفضل أن يتم بناء الهيكل المعماري للنظام من المكونات المعدة مسبقاً بقدر الإمكان.
- يفضل استخدام النماذج المرئية Visual Models لتمثيل المنظورات المختلفة للنظام.
- يجب التحقق من جودة البرمجيات المنتجة ومطابقتها لمقاييس الجودة الخاصة بالمؤسسة.
- يجب التحكم في التغيرات التي تحدث للبرمجيات من خلال إجراءات وأدوات إدارة التغيير.

١-٢-٣ التشييد المبني على النموذج (Model Driven Architecture (MDA

قدمت مجموعة إدارة الكائنات (Object Management Group (OMG هذا الأسلوب في محاولة للانتقال بلغة النمذجة الموحدة UML من مرحلة المواصفات الرسمية Formal Specifications إلى مرحلة المواصفات التنفيذية Executable Specifications، أو مرحلة إنتاج البرامج (Programs Generation Maciaszek, 2005). تقوم هذه الفكرة على مبدأ الاستفادة بالمعايير القياسية التي أسستها مجموعة OMG وكذلك الاستفادة من التقنيات الحديثة في تحويل نماذج المواصفات UML Models إلى أكواد قابلة للتنفيذ Executable Codes. يعتبر هذا الأسلوب نوعاً من أنواع النماذج التحويلية Transformational Models التي تقوم بتطوير النظام من خلال مجموعة متتالية من التحويلات للنماذج الرسمية، التي تصف متطلبات النظام، إلى مجموعة من البرامج التنفيذية، ويتم ذلك مروراً بمرحلة التصميم التفصيلية. يتم تدقيق كل خطوة تحويلية للتأكد من صحة مخرجاتها وأنها تمثل تحويلاً حقيقياً للمدخلات.

يعتبر هذا الأسلوب مدخلاً حقيقياً لميكنة عمليات التحويل من المواصفات إلى إنتاج البرامج، مع الأخذ في الاعتبار أن هناك بعض التحويلات يجب أن تتم يدوياً. توجد الآن مجموعة من أدوات هندسة البرمجيات المدعومة بالحاسب CASE هي تقوم بتنفيذ تلك التحويلات. تقوم هذه الأدوات بإنشاء وتخزين النماذج الرسمية التي تصف مكونات النظام، وبعد ذلك تقوم بتحويلها تدريجياً من خلال مجموعة من النماذج التحويلية. وعلى هذا فإن هذه الأدوات تدعم أسلوب التطوير المتدرج مع الزيادة. يوضح شكل (١-١٣) دورة حياة تطوير النظم باستخدام أسلوب التشييد المبني على النموذج.



شكل (١-١٣) تطوير النظم باستخدام أسلوب التشييد المبني على النموذج

يفصل هذا الأسلوب بين الشكل الرسمي للمواصفات وبين الشكل غير الرسمي للمتطلبات، وهو بذلك يقوم باستبعاد الشكل غير الرسمي للمتطلبات من عمليات التحويل، ويبدأ بنماذج المواصفات الرسمية التي يمكن قراءتها آلياً عن طريق الحاسب الآلي. تعتبر هذه النماذج غير مرتبطة ببيئة تشغيل معينة سواء من ناحية العتاد Hardware أو من ناحية البرمجيات Software. وبعد بدء عمليات التحويل تنشأ نماذج التصميم التفصيلية التي تعتمد على بيئة التشغيل. وإذا كان النظام المطور يعمل من خلال بيئات تشغيل متعددة، فإنه يتم توليد نماذج تفصيلية خاصة بكل بيئة تشغيل، وبعد ذلك يتم الربط بينها باستخدام مجموعات ربط تسمى «جسور التشغيل الداخلي Interoperability Bridges».

جدير بالذكر أن هذا الأسلوب للتطوير المبني على النموذج يعطى الأمل في إمكانية تطبيق تقنية التطوير بالمكونات التي يمكن من خلالها تطوير النظام بالكامل باستخدام المكونات الجاهزة أو المعدة مسبقاً. تكون هذه المكونات في شكل نماذج أو برامج يتم إعادة استخدامها لإنتاج العديد من البرمجيات (Pressman, 2001).

الفصل الثانى

نمذجة البرمجيات

Software Modeling

تتطلب عملية تطوير مشاريع البرمجيات قدراً كبيراً من التفاعل والاتصال بين أفراد فرق العمل القائمين على تطوير المشروع فيما بينهم من جهة، وبينهم وبين مستخدمى النظام من جهة أخرى. ولذلك فإنه من الضرورى أن يتم استخدام لغة أو وسيلة محددة لتسهيل عملية الاتصال والتفاعل بين جميع الأفراد المساهمين فى المشروع. يجب أن تكون اللغة المستخدمة لغة عامة ومفهومة لكل المشاركين فى المشروع، وأن تكون لغة قياسية، ومن الأفضل أن تكون ملائمة لعدد كبير من عمليات تطوير المشروع، وذلك لضمان اتساق وانسياب عملية التنقل من مرحلة إلى مرحلة أخرى فى أثناء عمليات التطوير باستخدام الأسلوب نفسه تقريباً.

يعتبر أسلوب النمذجة Modeling من الوسائل المناسبة لتوصيف عمليات تطوير البرمجيات واستخدامه بوصفه وسيلة للتفاهم والاتصال بين فرق العمل وذلك لاعتماده على النماذج المرئية بشكل أساسى لوصف المفاهيم والعمليات للنظام البرمجى. ويمكن أن ترفق مجموعة من النماذج النصية مع النماذج المرئية لوصف محتويات أو تفاصيل النماذج المرئية، بالإضافة إلى أنها تقوم بإضافة معلومات أخرى تعزز عملية فهم نظام الأعمال. ويمكننا القول إن عملية تحديد متطلبات النظام، التى تتم فى بداية عملية التطوير، وكذلك عملية توثيق البرامج المنتجة فى نهاية عملية التطوير، تتم عادة فى شكل نماذج نصية. وفى المقابل فإن المراحل المتوسطة فى أثناء عمليات تطوير النظام، والتحليل والتصميم، تتم عادة باستخدام النماذج المرئية. وعلى هذا فإن عمليات توصيف البرمجيات تتكون فى الغالب من مجموعة من النماذج المرئية Visual Models مصحوبة ببعض النماذج النصية Text Models، وذلك بهدف وصف عمليات التطوير المختلفة للنظام من عدة منظورات مختلفة Viewpoints. يحتوى النموذج غالباً على مخطط Diagram أو أكثر وأى معلومات أخرى توضيحية. يمثل المخطط وصفاً لإحدى وظائف أو مكونات النظام من وجهة نظر معينة وذلك بشكل مرئى باستخدام الرموز والرسومات التخطيطية. يجب تخزين تلك النماذج المرئية والنصية فى المستودع العام للنظام Repository ويتم استخدامه فيما بعد من جميع أفراد فريق التطوير للمشروع. هناك العديد من الأدوات التى تدعم هذه العملية يطلق

عليها اسم «أدوات هندسة البرمجيات المدعومة بالحاسب CASE Tools». سوف نستعرض الآن في هذا الفصل، بشكل سريع، أسلوبين فقط من أساليب النمذجة الشائعة الاستخدام في تطوير النظم. الأول هو أسلوب قديم مازال يستخدم حتى الآن وهو «النمذجة الهيكلية Structured Modeling» وهو يناسب الأنظمة التقليدية التي تم إنتاجها في السابق وتستخدم أسلوب البرمجة الهيكلية. والثاني هو أسلوب حديث يسمى «لغة النمذجة الموحدة Unified Modeling Language (UML)» ويتم استخدامه الآن بشكل واسع وهو يلائم النظم الكبيرة والمعقدة التي تستخدم أسلوب البرمجة الموجهة بالكائنات.

٢-١ النمذجة الهيكلية Structured Modeling

لقد ظهر أسلوب النمذجة الهيكلية في أواخر السبعينيات مواكباً لأسلوب البرمجة الهيكلية Structured Programming. تتسم النمذجة الهيكلية بأسلوب التصميم الهرمي من أعلى إلى أسفل Top-Down وهو أسلوب يقوم بتقسيم أو تجزئة النظام إلى مجموعة من الوظائف الفرعية المتفاعلة. تتم عملية التجزئة على عدة مستويات من حيث الإيجاز والتفصيل. فيكون الإيجاز في المستوى الأعلى Top وتكون التفاصيل في المستويات التالية إلى أسفل Down. يستخدم هذا الأسلوب في نمذجة متطلبات النظام واستخدامها لتعزيز عملية الفهم للنظام والتفاعل بين أفراد فريق التطوير. تعتمد النمذجة الهيكلية على توصيف عمليات النظام باعتبارها مدخلاً لتوصيف المهام الوظيفية للنظام. لقد أثبت هذا الأسلوب نجاحاً كبيراً في الفترة السابقة في أثناء استخدامه في تحليل الأنظمة الكبيرة، وهو يتميز باستخدام رموز وتخطيطات وتدوينات قياسية، بالإضافة إلى أنه يضمن إنجاز عملية توثيق النظام بشكل قياسي يساهم في فهم مكونات ومهام النظام بشكل جيد لجميع الأفراد القائمين على تطوير النظام وصيانته فيما بعد. يعاني أسلوب النمذجة الهيكلية من عدة نقاط ضعف، أهمها عدم المقدرة على توصيف المتطلبات غير الوظيفية للنظام، وعدم ملائمة أسلوب التطوير المبني على الكائنات Object-Oriented. هناك بعض أدوات هندسة البرمجيات المدعومة بالحاسب CASE التي تقوم بدعم عمليات نمذجة النظم باستخدام أسلوب النمذجة الهيكلية سيتم عرضها في الفصل الثالث.

تقدم النمذجة الهيكلية مجموعة من النماذج المرئية التي تستخدم في أثناء عمليات تحليل وتصميم الأنظمة. يمكننا تصنيف تلك النماذج في شكل مجموعتين رئيسيتين،

تمثل المجموعة الأولى مجموعة النماذج التى تستخدم فى نمذجة العمليات Process Modeling، وتمثل المجموعة الثانية النماذج التى تستخدم فى نمذجة البيانات Data Modeling، وسيتم وصفهما من خلال النماذج التالية:

- نماذج مخططات تدفق البيانات DFDs.
- نموذج مخطط العلاقة بين الكائنات ERD.
- وسوف نستعرضها الآن بشئ من التفصيل.

١-١-٢ مخططات تدفق البيانات (Data Flow Diagrams (DFDs

تعتبر مخططات تدفق البيانات DFDs هى أحد أهم النماذج المستخدمة فى الفترة السابقة لتوصيف المتطلبات الوظيفية لنظم الأعمال. يعتمد أسلوب إنشاء مخطط تدفق البيانات على مفهوم التفكيك الوظيفي Functional Decomposition. يوضح شكل (١-٢) مفهوم التفكيك الوظيفي من خلال تعريف العمليات الوظيفية للنظام فى شكل هرمى أو تدريجى من أعلى إلى أسفل Top-Down. يبدأ تعريف النظام من أعلى بمخطط السياق Context Diagram الذى يصف النظام بشكل عام وتفاعله مع البيئة المحيطة به. وبعد ذلك يتم وصف النظام من الداخل بالتدرج إلى أسفل فى عدة مستويات تفصيلية للعمليات الوظيفية للنظام. تبدأ هذه المستويات بالمستوى «صفر» الذى يصف العمليات الأساسية للنظام ويطلق عليه اسم «مخطط المنظور العام Level 0 Overview Diagram». يمكننا وصف تفاصيل كل عملية أساسية فى المستوى «صفر» من خلال مخططات تفصيلية فى المستوى التالى «واحد» ويطلق عليها اسم «مخططات تفصيلية Level 1 Details Diagrams». تأخذ كل عملية تفصيلية رقم مسلسل ينطلق من رقم العملية الأساسية لها مثل «عملية ١-١» و«عملية ٢-١» وهكذا. وإذا كانت إحدى العمليات التفصيلية فى المستوى «واحد» تحتاج إلى تفاصيل أكثر، يتم وصفها فى شكل مخطط تفصيلي فى المستوى التالى «اثنين» وتأخذ كل عملية تفصيلية رقما ينطلق من رقم العملية الأم لها مثل «عملية ٢-١-١» وهكذا لباقي العمليات التفصيلية فى جميع المستويات التالية.

العناصر الأساسية لمخططات تدفق البيانات:

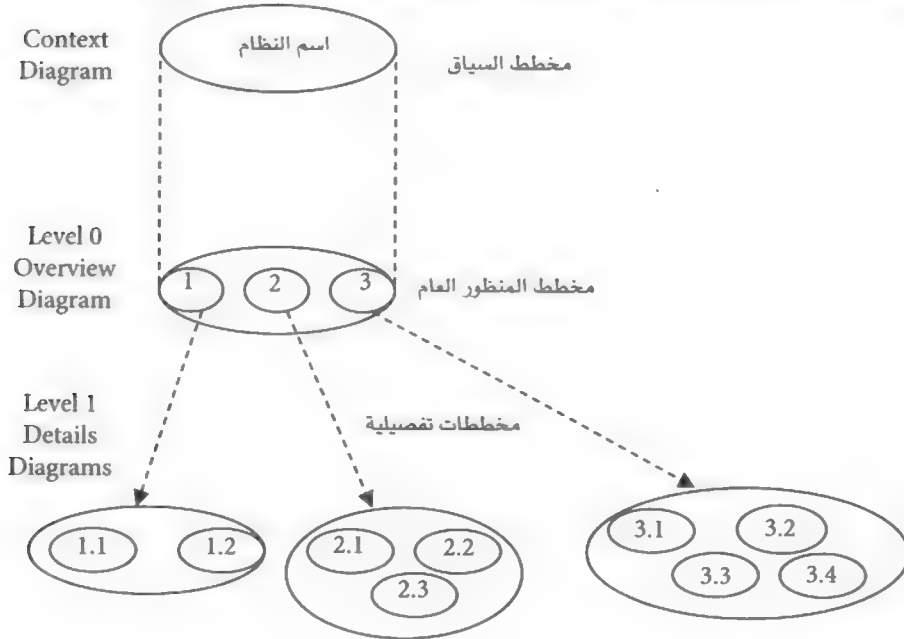
تحتوى مخططات تدفق البيانات على مجموعة من العناصر الأساسية التى تستخدم لوصف عمليات النظام فى المستويات المختلفة وهى (السماك، ٢٠٠٣):

١- الكائنات الخارجية External Entities

تمثل مجموعة الأشخاص والمؤسسات والأقسام والأنظمة الأخرى التي تتفاعل مع النظام من الخارج وتمده بالبيانات وتستقبل منه مخرجات.

٢- تدفق البيانات Data Flow

تمثل عملية نقل أو سريان البيانات من عنصر إلى عنصر آخر داخل مخطط تدفق البيانات DFD في شكل أسهم في اتجاه معين يوضح اتجاه سريان البيانات.



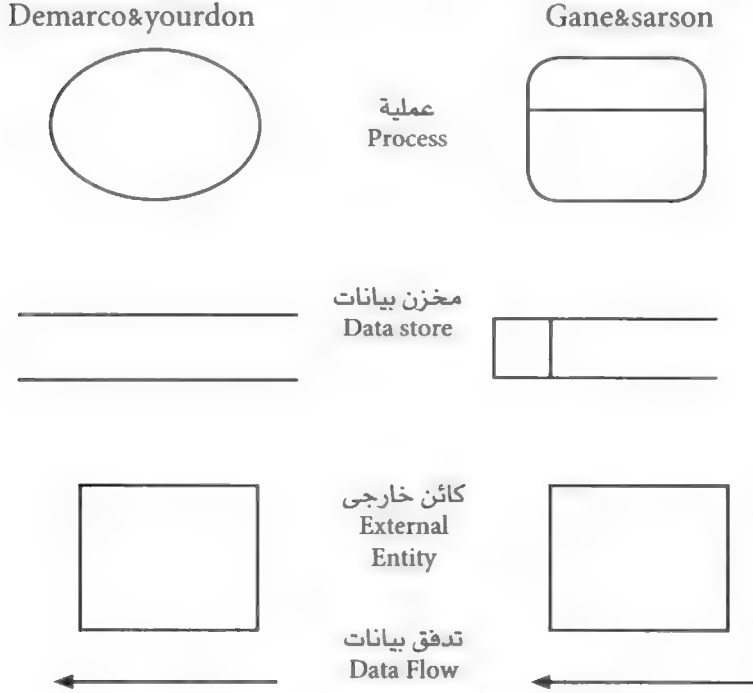
شكل (١-٢) التفكيك الوظيفي في مخططات DFDs

٣- العمليات Processes

تمثل مجموعة الأنشطة التي تقوم بمعالجة البيانات في النظام للحصول على المخرجات. هناك نوعان من العمليات: عمليات أولية أو أساسية، وعمليات فرعية أو تفصيلية للعمليات الأساسية.

٤- مخازن البيانات Data Stores:

هو مستودع يحتوى على بيانات كائنات النظام وكذلك تدفقات البيانات. يوضح شكل (٢-٢) الرموز المستخدمة لكل عنصر من عناصر مخططات تدفق البيانات فى كل من أسلوب "Yourdon & DeMarco" وأسلوب "Gane & Sarson" لنمذجة النظم الهيكلية (George et al., 2002).



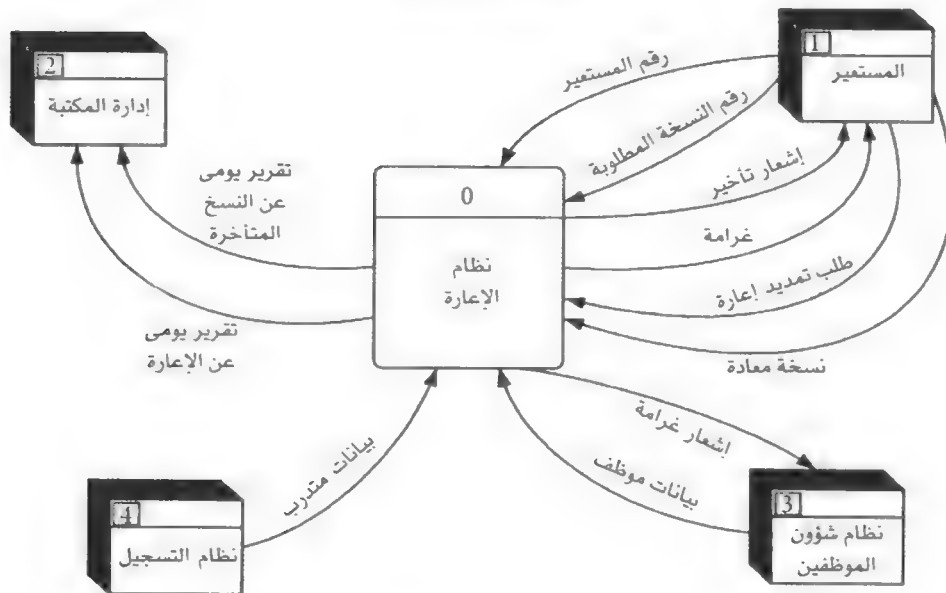
شكل (٢-٢) الرموز المستخدمة فى مخططات تدفق البيانات

٢-١-١-٢ مخطط السياق Context Diagram:

الفرض من مخطط السياق هو تحديد موقع نظام الأعمال داخل البيئة المحيطة به. يقوم المخطط أيضاً بتعريف حدود النظام وتعريف تفاعل النظام مع الكائنات الخارجية له من خلال مجموعة التدفقات القادمة منها والرسلة إليها، وهى تمثل مدخلات ومخرجات النظام. يتكون مخطط السياق للنظام من عملية واحدة تمثل

منظراً عاماً للنظام من الخارج وتفاعله مع بيئة العمل المحيطة به من خلال التدفقات بينه وبين الكائنات الخارجية له. يوضح شكل (٢-٣) مخطط السياق لنظام الإعارة بمكتبة معهد الإدارة العامة. يمثل كل من المستعير ونظام التسجيل ونظام شؤون الموظفين وإدارة المكتبة، الكائنات الخارجية للنظام أى أنها تمثل بيئة النظام. وتمثل التدفقات مثل «رقم المستعير»، و«طلب تمديد إعارة»، و«بيانات موظف»، التفاعلات المختلفة بين الكائنات الخارجية والنظام. سيتم عرض جميع العمليات الموجودة داخل النظام من خلال مخطط تدفق البيانات DFD.

Context Diagram



شكل (٢-٣) مخطط السياق لنظام الإعارة

٢-١-١-٢ مخطط المنظور العام DFD Overview:

يتم توصيف ما يحدث داخل مخطط السياق عن طريق مخطط المنظور العام، وهو يعتبر «المستوى صفر» لمخططات تدفق البيانات DFD Level-0. يركز هذا المخطط على وصف العمليات الرئيسية للنظام وتدفق البيانات خلالها، وهو يحتوى على ما يلي:

- الكائنات الخارجية External Entities.

- العمليات الأساسية Main Processes.

- مخازن البيانات Data Stores.

- تدفقات البيانات Data Flows.

يوضح شكل (٢-٤) مخطط المنظور العام لنظام الإعارة بمكتبة معهد الإدارة العامة. يحتوى المخطط على أربع عمليات رئيسية هي:

١- «أنشئ ملف مستعير» وهي العملية المسؤولة عن إنشاء ملف المستعيرين بالمكتبة وذلك بناء على البيانات المأخوذة من نظام القبول والتسجيل (بيانات المتدربين والطلاب) ونظام شئون الموظفين (بيانات الأساتذة والموظفين).

٢- «نفذ إعارة» وهي العملية المسؤولة عن استقبال طلب الاستعارة من المستعير والتحقق من إمكانية تنفيذه، ثم تسجيل بيانات الإعارة في ملف الإعارة، وتحديث بيانات الكتاب المعار، وعدد الكتب المعارة للمستعير.

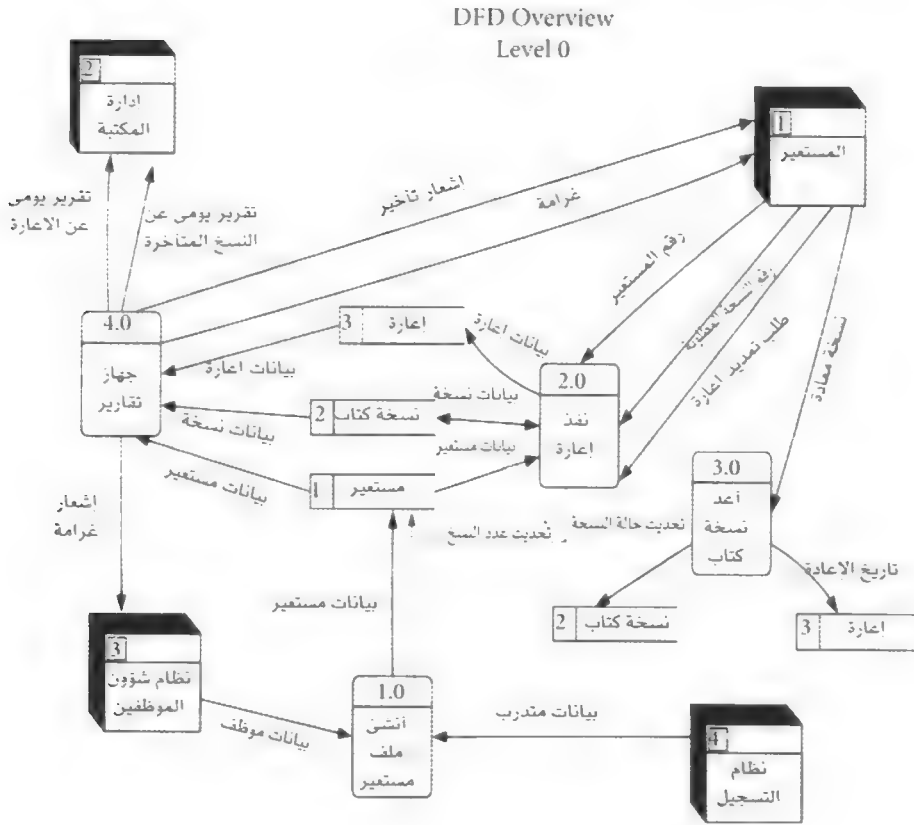
٣- «أعد نسخة كتاب» وهي العملية المسؤولة عن تسجيل بيانات إعادة نسخة الكتاب المعارة إلى المكتبة، وتحديث بيانات الإعارة، وحالة الكتاب، وعدد الكتب المعارة للمستعير.

٤- «جهز تقارير» وهي العملية المسؤولة عن إعداد جميع التقارير المطلوبة لإدارة النظام من حيث تصميم الشكل المطلوب للتقرير وتكوين جملة الاستعلام المناسبة لتنفيذ التقرير. بالإضافة إلى إصدار إشعارات التأخير والغرامات للمستعيرين المتأخرين عن موعد إعادة الكتب المعارة لديهم أو فقدانها.

ويحتوى المخطط أيضاً على ثلاثة مخازن للبيانات هي «مستعير»، و«نسخة كتاب»، و«إعارة». نلاحظ في الشكل أن التدفقات الخارجية للنظام هي التدفقات الخارجية الموجودة نفسها بمخطط السياق، ولكن هناك تفاصيل أدق لتدفقات أخرى داخلية بين العمليات ومخازن البيانات.

يجب أن يراعى عند تصميم مخطط المنظور العام لتدفق البيانات أن تكون الاعتمادية بين العمليات الرئيسية التي يتضمنها أقل ما يمكن، بحيث لا تعتمد عملية ما على عملية أخرى عند تنفيذها. فمن المفترض أن تكون الوسيلة الطبيعية لبدء تشغيل

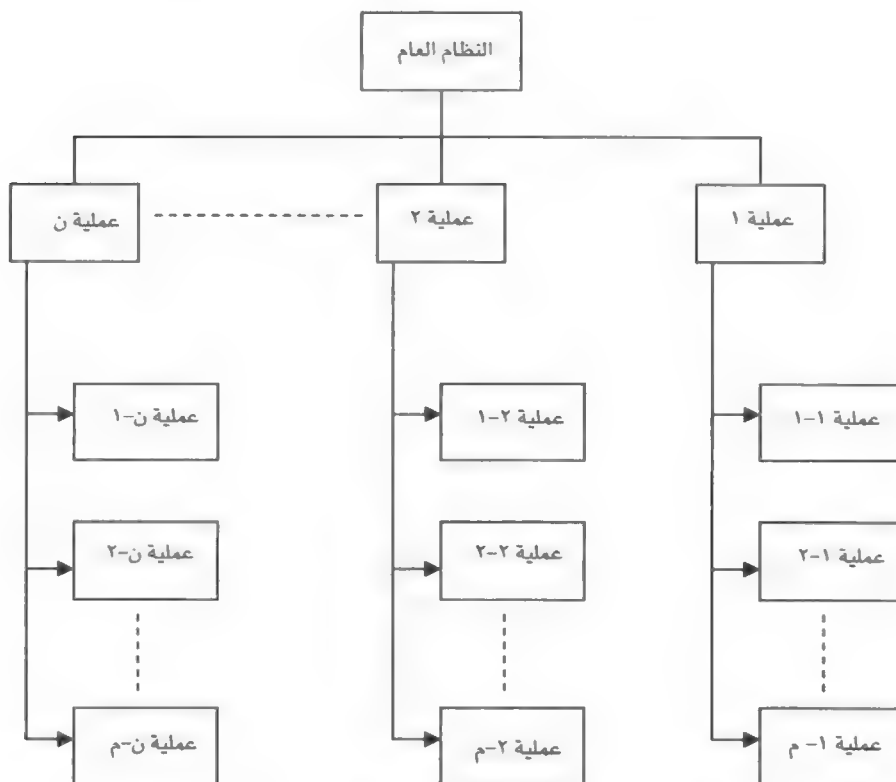
أي عملية رئيسية من خلال الكائنات الخارجية التي تتفاعل مع النظام أو من خلال ميقات زمني محدد. وللتغلب على مشكلة الاعتمادية التي يمكن أن تنشأ بين العمليات الرئيسية بالنظام، يتم استخدام مخازن البيانات كوسيط طبيعي لتبادل المعلومات. فيمكن لأي عملية أن تستقبل مدخلاتها وتعالجها ثم تقوم بتسجيل مخرجاتها في مخازن البيانات، ومن ثم يمكن لأي عملية أخرى بالنظام استخدام تلك المعلومات المخزنة باعتبارها مدخلات طبيعية لها.



شكل (٢-٤) مخطط المنظور العام لنظام الإعارة

٣-١-١-٢ المخططات التفصيلية Detailed DFDs:

يتم توصيف مجموعة الأنشطة التي تتضمنها العمليات الأساسية للنظام من خلال مجموعة من المخططات التفصيلية لمكونات مخطط المنظور العام، وذلك في عدة مستويات داخلية إلى أسفل. يتم تفكيك العملية الأساسية في مخطط المنظور العام إلى مجموعة من العمليات الفرعية المترابطة في شكل مخطط تدفق بيانات منفصل لهذه العملية في مستوى تال. تبدأ المخططات التفصيلية بالمستوى واحد DFD Level-1. وإذا كانت إحدى هذه العمليات الفرعية تحتوي على تفاصيل أخرى، فإنه يتم تفكيكها في شكل مخطط تدفق بيانات منفصل في المستوى الثاني DFD level-2، وهكذا. يعرف هذا الأسلوب بأسلوب «التفكيك الوظيفي Functional Decomposition»، كما هو موضح بالشكل (٥-٢).



شكل (٥-٢) التفكيك الوظيفي للعمليات

ويوضح شكل (٢-٦) مخطط تدفق البيانات، المستوى واحد DFD Level-1 للعملية رقم ٢ «نقد إعارة» في مخطط المنظور العام لنظام الإعارة الموضح بالشكل (٢-٤). نلاحظ من الشكل أن العمليات الفرعية تتبع نظام ترقيم يبدأ برقم العملية الأساسية ثم الرقم الفرعي مثل ٢-١، ٢-٢، ٢-٣ وهكذا. تحتوى العملية الأساسية «نقد إعارة» على أربع عمليات تفصيلية هي:

٢-١ «حدد المستعير» وهى المسئولة عن استخراج بيانات المستعير من ملف المستعير بناء على البيان المدخل لها وهو رقم المستعير.

٢-٢ «دقق عملية الإعارة» وهى المسئولة عن تدقيق بيانات الإعارة المطلوبة، فتبدأ بتحديد نوع العملية المطلوبة «إعارة جديدة/تجديد إعارة»، وبعد ذلك يتم التحقق من حالة نسخة الكتاب المطلوبة للإعارة «متاحة/معاراة» وذلك باستخدام بيان رقم نسخة الكتاب. وإذا كانت نسخة الكتاب متاحة للإعارة فإنه يتم التأكد من أحقية المستعير لإتمام عملية الإعارة وذلك من خلال معرفة عدد الكتب المعارة للمستعير ومقارنتها بالعدد المسموح به، والتأكد من عدم وجود محاذير على المستعير نفسه كأن يكون لديه كتب متأخرة أو مفقودة (أى أنه موجود بالقائمة السوداء).

٢-٣ «سجل الإعارة» وهى المسئولة عن تنفيذ عملية الإعارة من عدمها بناء على البيانات المقدمة لها من العملية السابقة، فيتم تسجيل بيانات الإعارة فى ملف الإعارة وتحديث عدد النسخ المعارة للمستعير وتحديث حالة نسخة الكتاب المعار.

٢-٤ «مدد الإعارة» وهى المسئولة عن تحديث بيانات الإعارة (تاريخ الاسترجاع لنسخة الكتاب المعار) فى حالة طلب المستعير تمديد فترة الإعارة لنسخة الكتاب المعار لديه.

ونلاحظ أيضاً أن التدفقات الخارجية للمخطط التفصيلي هى نفسها التى تصف حدود العملية الأساسية «نقد إعارة» فى مخطط المنظور العام وهى نفسها الموجودة بمخطط السياق، وذلك لضمان تحقيق الاتساق التام بين مخططات وصف النظام فى المستويات المختلفة. بالإضافة إلى وجود تدفقات أخرى داخلية بين العمليات التفصيلية وبين مخازن البيانات. ومن الممكن أن تكون هناك عملية تفصيلية فى هذا المستوى Level 1 تحتاج إلى تفاصيل أكثر. وهنا يجب أن يتم وصفها من خلال مخطط منفصل فى المستوى الثانى Level 2 ويتضمن عمليات تبدأ بترقيم العملية السابقة فى المستويين السابقين مثل ٢-١-١، ٢-١-٢، وهكذا.

شكل (٢-٦) مخطط تدفق البيانات التفصيلي لعملية «نقد إعاره»

٢-١-٢ وصف منطق العمليات Process Logic Description:

تتطلب عملية توصيف المتطلبات الوظيفية للنظام أن يتم توصيف منطق العمليات الفرعية للنظام في شكل نماذج نصية، ويتم إرفاقها مع مخططات تدفق البيانات للنظام. الهدف من هذه النماذج هو وصف إجراءات العمل لكل عملية بشكل منطقي لا يعتمد على الأدوات أو التقنية التي ستقوم بتنفيذها. هناك عدة طرق تستخدم لوصف منطق العمليات منها:

- الإنجليزية الهيكلية Structured English أو العربية الهيكلية.

- مخططات العمل Action Diagrams.

- جداول القرارات Decision Tables.

- شجرة القرارات Decision Tree.

ومن الضروري أن تتسم عملية توصيف منطق العمليات بالأسلوب المنطقي فقط ولا تستخدم أسلوب لغة برمجة معينة ولكن تتضمن المفاهيم الأساسية للبرمجة الهيكلية. وتعتبر هذه النماذج النصية أداة تفاهم مشتركة بين أفراد فريق التطوير من المحللين والمصممين والمبرمجين.

٢-١-٢-١ الإنجليزية الهيكلية Structured English:

يفضل أغلب المحللين استخدام أسلوب الإنجليزية (أو العربية) الهيكلية لوصف منطق العمليات وذلك لتمييزها بالجمع بين أسلوب اللغة الطبيعية والأسلوب الهيكلي الذي يلائم المنطق في أثناء صياغة الجمل التي تصف خطوات تنفيذ كل عملية. يتسم هذا الأسلوب بعدة مميزات منها:

- سهولة وصف الإجراءات وذلك باستخدام لغة طبيعية.

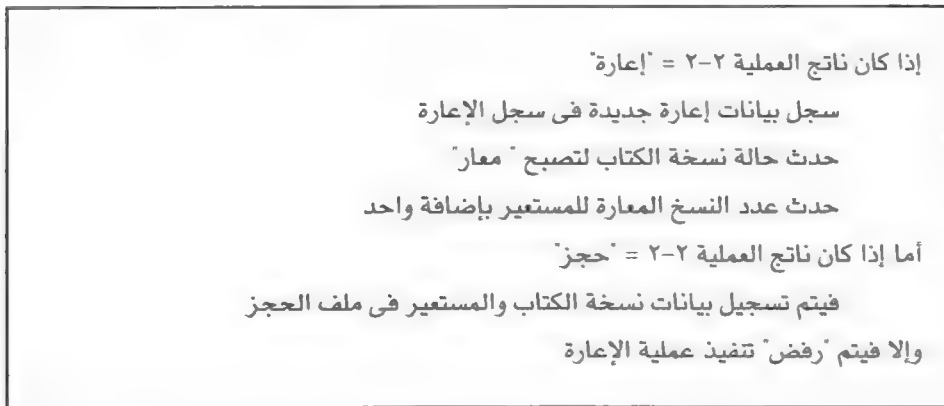
- عدم التقيد باستخدام أسلوب لغة برمجة معينة.

- المرونة في استخدام الهياكل البرمجية الأساسية مثل التدفقات والتفرعات والتكرارات.

- تسهيل عملية التفاهم والاتصال بين فرق العمل بالمشروع (تحليل، تصميم، برمجة).

- سهولة صيانة تفاصيل العمليات.

يفضل بعض أفراد التحليل استخدام أسلوب مخططات العمل أثناء وصف منطق العمليات وذلك لقربها من الأسلوب البرمجي ولكن دون التقيد بلغة برمجة معينة. يفضل استخدام هذا الأسلوب في حالة اشتراك أفراد فريق العمل في أكثر من نشاط (تحليل، تصميم، برمجة) في أثناء عملية التطوير. ولكن يعيب هذا الأسلوب صعوبة صيانة منطق العمليات فيما بعد خاصة من جانب أفراد آخرين التحقوا بالعمل في المشروع. ولذلك فإنه من الأفضل استخدام أسلوب الإنجليزية الهيكلية مع مراعاة تغليب الطابع الهيكلي على الطابع الإنشائي للجمل المستخدمة لوصف منطق العملية. يؤدي هذا الأسلوب إلى الجمع بين مميزات الأسلوبين من حيث سهولة الفهم واستخدام التركيبات المنطقية المشابهة للتركيبات البرمجية. ومع ذلك فهناك بعض أفراد فريق التحليل يفضلون استخدام أسلوب مخطط التدفق Flow Chart لوصف منطق العمليات بشكل يجمع بين الأسلوب التخطيطي والأسلوب البرمجي. يوضح شكل (٧-٢) مثالاً على استخدام أسلوب العربية الهيكلية لوصف منطق العملية الفرعية «سجل الإعارة» في نظام الإعارة.



شكل (٧-٢) مثال على استخدام أسلوب العربية الهيكلية

وفي حالة احتواء العملية على عدد كبير من الشروط والأفعال مما يؤدي إلى كثرة التفريعات، ومن ثم صعوبة في الوصف أو الفهم، فإنه يفضل استخدام أسلوب جداول القرارات أو شجرة القرارات. يتميز هذان الأسلوبان بدقة وسهولة التعامل مع التفريعات الناتجة عن تعدد الشروط وتداخلها وذلك ضمن قواعد العمل التي تحكم إجراءات عملية معينة تتصف بالتعقيد أو صعوبة التركيب. وفيما يلي عرض بسيط

لاستخدام الأسلوبين في وصف منطق عملية «دقق عملية الإعارة» في نظام الإعارة بمكتبة معهد الإدارة العامة.

٢-٢-١-٢ جداول القرارات Decision Tables،

يتم أولاً تحديد مجموعة الشروط والأفعال الخاصة بعملية التدقيق وهي كما يلي

الشروط:

- ١- هل المستعير موجود بالقائمة السوداء نتيجة فقدان كتب معارة لديه أو كثرة التأخير في استرجاع الكتب المعارة؟
- ٢- هل عدد الكتب المعارة حالياً للمستعير أقل من الحد الأقصى المسموح به؟
- ٣- هل نسخة الكتاب المطلوب إعارتها متاحة للإعارة أو لا؟

الأفعال:

- ١- إعارة (تنفيذ عملية الإعارة للمستعير).
 - ٢- حجز (حجز نسخة الكتاب للمستعير في حالة عدم توافر أى نسخة الآن للإعارة أو في حالة وصول عدد الكتب المعارة للمستعير للحد الأقصى).
 - ٣- رفض (رفض تنفيذ عملية الإعارة للمستعير نظراً لوجوده ضمن القائمة السوداء).
- نقوم الآن بتمثيل تلك المعلومات من خلال جدول القرارات الذي يأخذ شكل الجداول المنطقية (شكل (٢-٨)) كما يلي:

الأفعال	شروط ٣	شروط ٢	شروط ١
رفض	لا	لا	لا
رفض	لا	لا	نعم
حجز	لا	نعم	لا
رفض	لا	نعم	نعم
حجز	نعم	لا	لا
رفض	نعم	لا	نعم
إعارة	نعم	نعم	لا
رفض	نعم	نعم	نعم

شكل (٢-٨) الشكل الأولي لجدول القرارات لعملية «دقق عملية الإعارة»

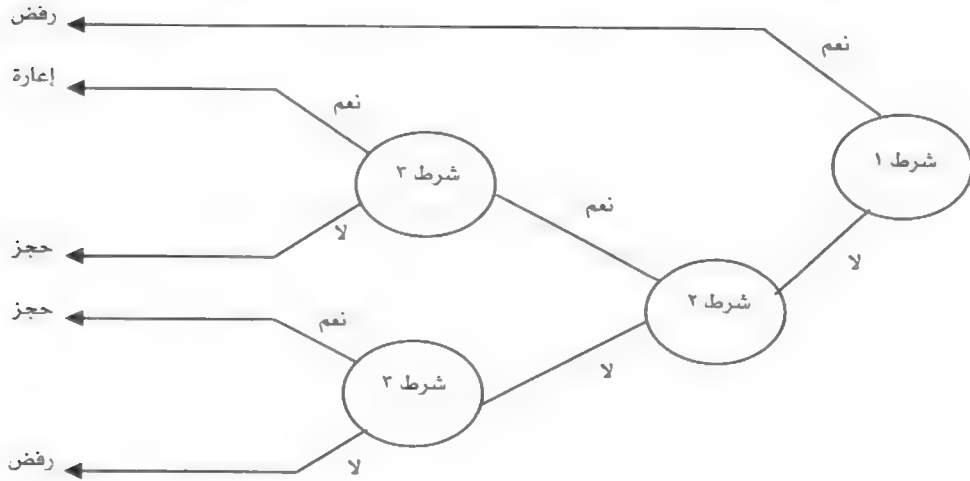
نلاحظ من الجدول أنه يتم تنفيذ عملية الإعارة فى حالة واحدة فقط، والحجز فى حالتين، والرفض فى باقى الحالات. وبدراسة حالات الرفض نجد أن عددها خمس حالات، منها أربع حالات تتضمن الشرط ١ عندما تكون قيمته تساوى «نعم». ولذلك فسوف نقوم بتمثيل الحالات الأربع بحالة واحدة تحتوى على قيمة الشرط الأول فقط بغض النظر عن قيم الحالات الأخرى. يودى ذلك إلى إنتاج شكل مبسط لجدول القرارات كما هو مبين بالشكل (٩-٢).

الافعال	شرط ٢	شرط ٢	شرط ١
رفض	لا	لا	لا
رفض	-	-	نعم
حجز	لا	نعم	لا
حجز	نعم	لا	لا
إعارة	نعم	نعم	لا

شكل (٩-٢) الشكل المبسط لجدول القرارات لعملية «دقق عملية الإعارة»

٢-١-٢-٣ شجرة القرارات Decision Tree؛

يمكننا تمثيل المعلومات السابقة نفسها بأسلوب آخر يأخذ شكل شجرة المعلومات وهو ما يسمى بشجرة القرارات. يبدأ جذر الشجرة بالشرط الأول وتكون فروعها هي القيم المسموح بها للشرط، وتكون محصورة فى قيمتين فقط بالنسبة للشروط المنطقية هما (نعم/لا). وينطلق الشرط الثانى من نهاية فرعى الشرط الأول مما يودى إلى أربعة فروع. وينطلق الشرط الثالث من نهاية الفروع الأربعة السابقة، مما يودى إلى ثمانية أفرع، وهكذا لباقى الشروط حتى نصل إلى القرارات التى تمثل العملية. يمكننا أيضاً تبسيط شجرة القرارات بدمج الفروع التى تودى إلى القرار نفسه وتعتمد على شرط معين أو أكثر لاتخاذ القرار بغض النظر عن قيمة باقى الشروط. يوضح شكل (١٠-٢) الشكل المبسط لشجرة القرارات المقابل لجدول القرارات المبسط (شكل (٩-٢)).



شكل (١٠-٢) الشكل المبسط لشجرة القرارات لعملية «دقق عملية الإعارة»

٣-١-٢ مخطط العلاقة بين الكائنات (ERD) Entity Relationship Diagram

فى حين كانت تركيز مخططات تدفق البيانات DFDs على وصف عمليات نظام الأعمال، فإن مخططات العلاقة بين الكائنات ERDs تركيز على وصف كائنات نظام الأعمال وتمثيل العلاقات بينها، فيما يعرف بنمذجة بيانات النظام Data Modeling. يساهم أسلوب نمذجة بيانات النظام فى تنظيم وتوثيق جميع بيانات نظام الأعمال، ويعتبر هو المدخل الطبيعى لتصميم قواعد بيانات النظام. تتم نمذجة بيانات النظام بشكل مرئى من خلال تمثيل جميع كائنات النظام Entities بشكل منطقى وتمثيل العلاقات المنطقية بينها Relationships بغض النظر عن التقنية المستخدمة لتنفيذها. يتضمن نموذج ERD ما يلى (السماك، ٢٠٠٣):

١-٣-١-٢ كائنات النظام System Entities

يمثل الكائن Entity أى شىء له دور فعال فى نظام الأعمال ويتطلب تسجيل بياناته فى مخازن بيانات النظام. فيما يلى أمثلة لبعض الكائنات الشائعة الاستخدام فى نظم الأعمال.

- شخص (عميل، طالب، مدرس، موظف،.....).
- شئ (كتاب، سلعة، جهاز، سيارة، عقار،.....).
- مكان (منطقة، مخزن، فرع، مبنى،.....).
- حدث (بيع، شراء، تسجيل، حجز، طلب،.....).
- مفهوم (حساب، مخزون، منهج دراسى، اعتماد مالى،).

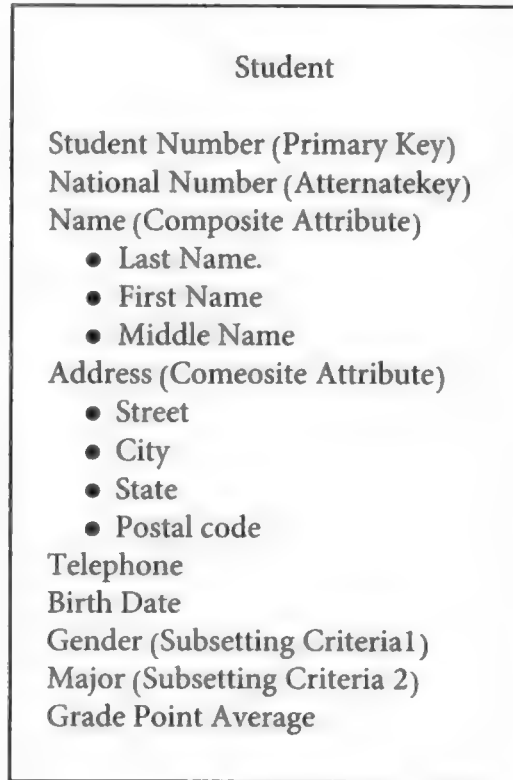
خصائص الكائنات Entities Attributes

تصف خصائص الكائن مجموعة الصفات أو السمات التى تميزه عن باقى الكائنات مثل الاسم، العنوان، الوظيفة، وغيرها. أحياناً يحتوى الكائن على خصائص مركبة، أى أنها تحتوى على خصائص أخرى فرعية مثل:

الاسم (الأول، المتوسط، الأخير).

العنوان (الدولة، المدينة، الرمز البريدى، صندوق البريد).

يحتوى كل كائن على خاصية أو أكثر تميز حالات الكائن Instances، أى أنها تأخذ قيمة وحيدة Unique Value لا يمكن أن تتكرر فى حالات أخرى للكائن. تستخدم هذه الخاصية لتحديد هوية الكائن أو ما يطلق عليه «مفتاح أساسى Primary Key». إذا كان هناك أكثر من خاصية يمكن أن تستخدم مفتاحاً أساسياً، فيتم اختيار أحدها ويكون الآخر مفتاحاً بديلاً Alternate Key يمكن استخدامه عند الضرورة فيما بعد أثناء صيانة أو توسعة نظام الأعمال. وهناك خصائص أخرى يتكرر حدوثها فى حالات عديدة للكائن وتكون غالباً مماثلة لقيم مفتاح أساسى فى كائن آخر، ويطلق عليها «مفاتيح خارجية Foreign Keys». يستخدم هذان النوعان من المفاتيح لتوصيف العلاقات المختلفة بين كائنات النظام. يوضح شكل (٢-١١) مثلاً لخصائص كائن «طالب».



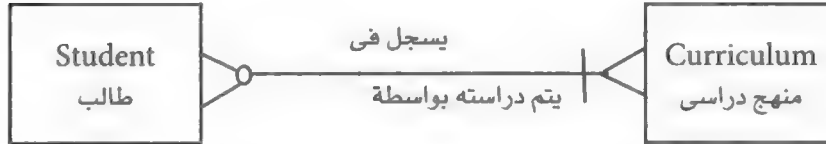
شكل (١١-٢) خصائص كائن «طالب»

٢-٣-١-٢ العلاقة بين الكائنات Entities Relationships:

العلاقة هي رابطة عمل طبيعية موجودة بين واحد وأكثر من الكائنات داخل النظام. من الممكن أن تمثل العلاقة حدثاً Event معيناً يربط بين الكائنات أو تمثل علاقة قرابة منطقية موجودة أصلاً بين الكائنات. فمثلاً تكون العلاقة المنطقية التي تربط بين المنهج الدراسي والطالب نتيجة حصول حدث معين وهو إجراء عملية التحاق الطالب بهذا المنهج الدراسي. وتكون العلاقة بين المدرس والمادة من خلال عملية التدريس أو التأهيل للتدريس. وتكون العلاقة بين المسافر ورحلة طيران معينة من خلال عملية الحجز، وهكذا. وكمثال للعلاقة بين الكائنات نتيجة القرابة الطبيعية بينها، نجد العلاقة الطبيعية بين العقار ومالك العقار، إذ إن لكل عقار مالكاً معيناً، والعلاقة بين الأب والتابعين له في الملف الصحي أو الاجتماعي، وهكذا.

١-٢-٣-١-٢ تعددية العلاقة Cardinality

من الطبيعي أن يكون لأي علاقة اتجاهان (اتجاه لكل طرف من طرفي العلاقة) ويجب تحديد نوع العلاقة لكل اتجاه. تبين تعددية العلاقة العدد الأكبر Maximum والعدد الأصغر Minimum لتكرار حدوث حالات أحد الكائنات المتعلقة بحدوث حالة كائن آخر مرتبط به. يجب تحديد التعددية في كلا الاتجاهين للعلاقة نظراً لأن كل العلاقات ثنائية الاتجاه. فمثلاً العلاقة بين الطالب والمنهج الدراسي تعتبر من النوع «واحد أو أكثر إلى صفر أو أكثر Many-To-Many»، فمن جهة كائن الطالب نجد أن الطالب الواحد يمكن أن يلتحق بمنهج دراسي أو أكثر. ومن جهة كائن المنهج الدراسي نجد أن المنهج الواحد يمكن أن يتم دراسته من قبل طالب أو أكثر أو لا يتم دراسته حتى الآن، كما هو مبين بالشكل (١٢-٢).



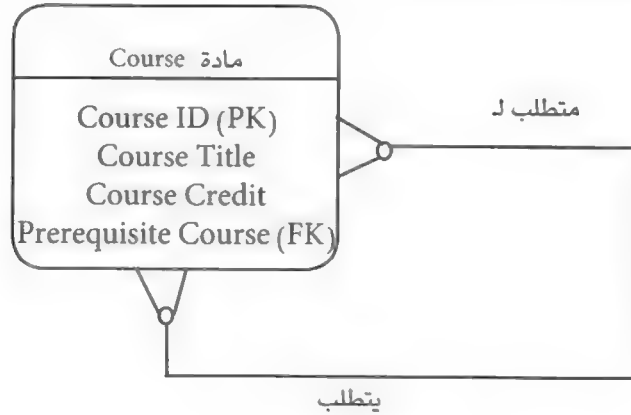
شكل (١٢-٢) تمثيل العلاقة بين الكائنات

١-٢-٣-٢-٢ درجة العلاقة Degree of Relationship

تمثل درجة العلاقة عدد الكائنات التي تشارك في العلاقة وهي ثلاثة أنواع:

١- علاقة ذاتية Recursive Relationship أو أحادية Unary

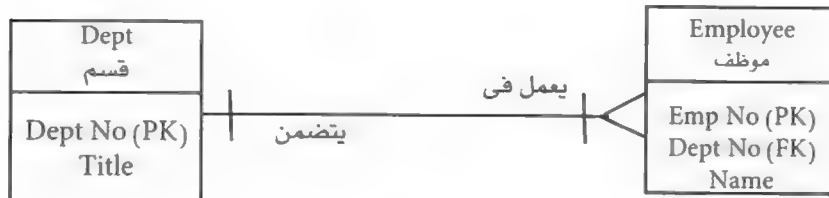
هي علاقة بين حالات Instances من الكائن نفسه، وفيها يحتوى الكائن على خاصية (حقل) ترتبط بخاصية أخرى موجودة كما في المثال التالي (شكل (١٢-٢)). فنجد أن المادة Course يمكن أن تتطلب إتمام دراسة مادة أخرى أو أكثر كمطلب أساسي لتدريسها. يتم وصف تلك العلاقة بين رقم المادة Course ID (المفتاح الأساسي للكائن) وبين رقم المادة المتطلبة Prerequisite Course (المفتاح الخارجي داخل الكائن نفسه). وبالمثل فإنه من الممكن أن تكون المادة متطلباً أساسياً لمادة أخرى أو أكثر. ولذلك فإن العلاقة تأخذ الشكل متعدد - إلى - متعدد ومن النوع «اختياري» في الاتجاهين.



شكل (١٣-٢) العلاقة الذاتية

٢- علاقة ثنائية Binary Relationship:

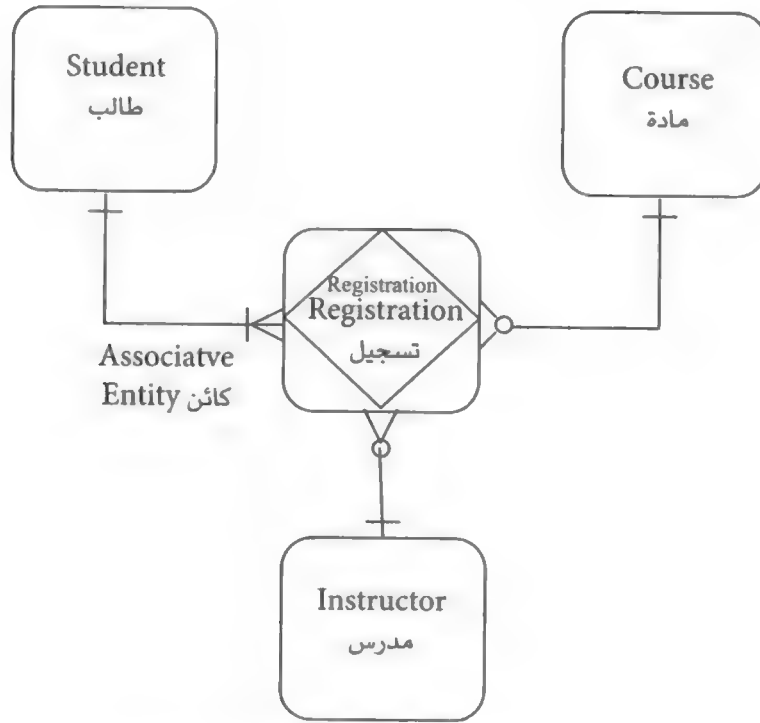
هى علاقة بين حالات كائنين، يتمثل الارتباط فيها من خلال خاصية (أو أكثر) مشتركة فى الكائنين وتكون تلك الخاصية هى المفتاح الأساسى فى أحد الكائنين وتكون المفتاح الخارجى فى الكائن الآخر المرتبط به. يوضح الشكل (١٤-٢) إحدى العلاقات الثنائية التى تمثل العلاقة بين الموظف Employee والقسم Dept الذى يعمل به. فنجد أن كل قسم يحتوى على مجموعة من الموظفين العاملين به، وأن كل موظف يجب أن يكون ملتحقاً بقسم معين. أى أن العلاقة من النوع واحد - إلى - متعدد من ناحية القسم باتجاه الموظف. ولذلك فقد تم إدراج رقم القسم ضمن خصائص الموظف باعتباره مفتاحاً خارجياً يرتبط بكائن القسم Dept ليصف عملية الربط بين الكائنين.



شكل (١٤-٢) العلاقة الثنائية

٣- علاقة متعددة N-ary Relationship

تعتبر العلاقات الثنائية من أوضح أنواع العلاقات بين الكائنات وأسهلها عند التطبيق من خلال أحد نظم إدارة قواعد البيانات المتاحة. ولكن كما ذكرنا من قبل فإن تمثيل العلاقات بين الكائنات يكون ترجمة حقيقية لطبيعة الارتباط الفعلى بين تلك الكائنات. وأحياناً يكون الارتباط ممثلاً من خلال علاقة مركبة بين أكثر من كائنين، كما فى المثال الموضح بالشكل (٢-١٥) الذى يوضح علاقة مركبة بين ثلاثة كائنات هى «الطالب» و«المدرس» و«المادة» وهى تحدث فعليا فى أثناء عملية التسجيل للطالب فى مادة معينة قبل بداية الفصل الدراسى فى نظام التسجيل بالجامعات بأسلوب الساعات المعتمدة. ونلاحظ أن تلك العلاقة تحتوى على خصائص معينة تصف عملية التسجيل نفسها مثل «تاريخ التسجيل» و«المعدل الفصلى». ويتم تمثيل تلك العلاقة المركبة من خلال كائن آخر يمثل عملية الارتباط بينهم يسمى «كائن الربط Associative Entity» وهو يستخدم للربط بين أكثر من كائن، وهو يحتوى على مفتاح أساسى مركب موروث من الكائنات الأخرى المرتبطة من خلاله. يشير كل جزء من المفتاح الأساسى المركب لكائن الربط إلى مفتاح أساسى لكائن من الكائنات المرتبطة، وهو يمثل أحد المفاتيح الخارجية لكائن الربط. ويتضمن أيضاً كائن الربط الخصائص الإضافية التى تخص العلاقة الناتجة عن الحدث وهو «التسجيل» مثل «تاريخ التسجيل» و«المعدل الفصلى». والجدير بالذكر هنا أنه لا ينبغى أن نستبدل تلك العلاقة الثلاثية الصريحة بمجموعة من العلاقات الثنائية البسيطة كحل للمشكلة. كأن يتم تمثيل العلاقة الثلاثية من خلال ثلاث علاقات ثنائية بين كل من «الطالب-المادة» و«الطالب-المدرس» و«المدرس-المادة». إذ إن تلك العلاقات هى بمنزلة علاقات افتراضية غير واقعية مطلقاً. فمثلاً لا توجد علاقة طبيعية بين الطالب والمادة إلا من خلال عملية التسجيل، وكذلك لا توجد علاقة طبيعية بين الطالب والمدرس إلا من خلال عملية التسجيل أو التدريس. وأيضاً لا توجد علاقة طبيعية بين المدرس والمادة إلا من خلال عملية التسجيل أو التدريس. ولذلك فإن عملية التسجيل أو التدريس هى العلاقة الطبيعية التى تربط بين الكائنات الثلاثة فى الوقت نفسه. وبالمثل نجد أن هذا المفهوم ينطبق على العديد من التطبيقات العملية مثل قيام الموظف بعملية حجز موعد لمريض فى عيادة معينة عند طبيب محدد، أو قيام موظف معين بتسجيل عملية حجز لمسافر فى رحلة طيران معينة، أو تكليف موظف بتنفيذ مهمة محددة فى مشروع معين يتطلب مهارة محددة، وغيرها من التطبيقات التى تحتوى على علاقات متعددة.



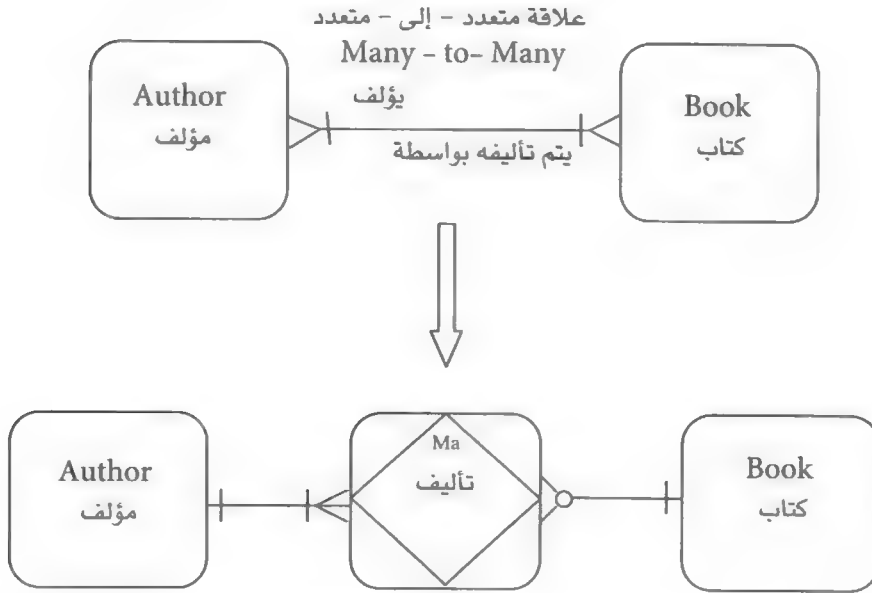
شكل (٢-١٥) علاقة متعددة وكائن الربط

يستخدم أيضاً أسلوب كائن الربط لحل مشكلة العلاقة المعقدة «متعدد - إلى - متعدد» الموجودة بين كائنين، وتحويلها إلى علاقتين بسيطتين من النوع «واحد - إلى - متعدد» من الناحيتين، كما هو موضح بالشكل (٢-١٦) الذي يمثل العلاقة بين المؤلف والكتاب. فطبيعة العلاقة أن المؤلف يمكن أن يؤلف كتاباً أو أكثر، وبطبيعة الحال فإن الكتاب الواحد يمكن أن يشارك في تأليفه مؤلف واحد أو أكثر. ولتبسيط تلك العلاقة المركبة، يتم تمثيل العلاقة في شكل كائن ربط «تأليف» يحتوى على بيانات الكتب التي شارك في تأليفها كل مؤلف، أو بمعنى آخر بيانات المؤلفين لكل كتاب. فنجد أن كائن الربط «تأليف» يحتوى على الخصائص التالية كبيانات أساسية للكائن:

- رقم المؤلف.

- رقم الكتاب.

بالإضافة إلى أي بيانات أخرى تخص عملية التأليف. ويكون المفتاح الأساسي للكائن «تأليف» هو مفتاحاً مركباً يتكون من كلتا الخاصيتين.

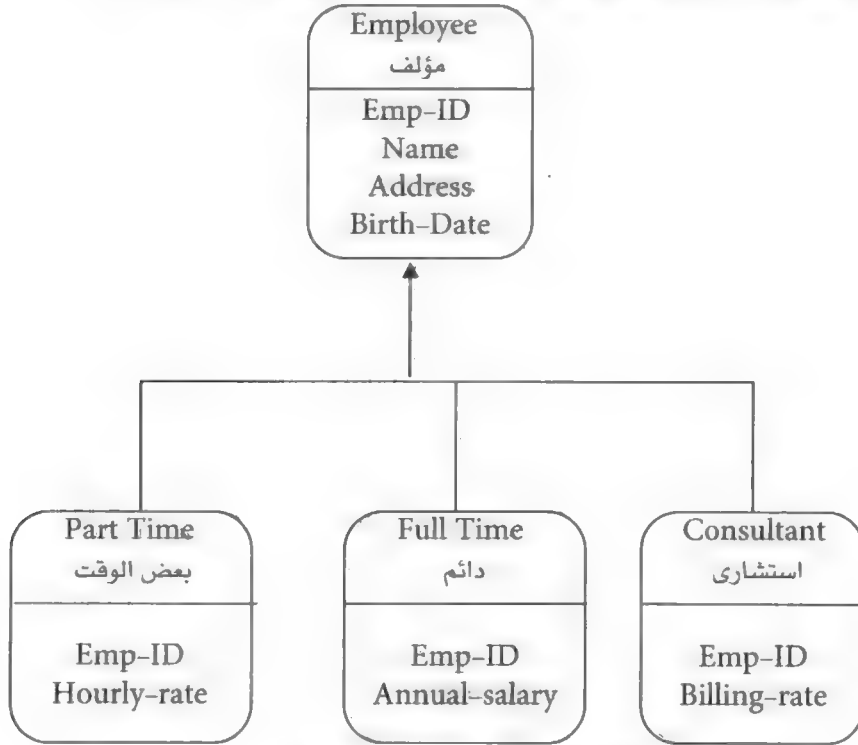


شكل (٢-١٦) استخدام كائن الربط Associative Entity

٤- التعميم Generalization

التعميم هو أسلوب يتم فيه تجميع الخصائص المشتركة لعدد من الكائنات في شكل نوع كائن منفصل يسمى نوع علوى Supertype. وتكون باقى الكائنات عبارة عن أنواع فرعية Subtype ترث الخصائص المشتركة من النوع العلوى Supertype وتضيف إليه خصائص أخرى تميز حالات كل نوع فرعى. ومثال ذلك يوضح الشكل (٢-١٧) تمثيل أنواع التوظيف المختلفة في بعض المؤسسات. فنجد أنه بغض النظر عن طبيعة الوظيفة فإن لكل موظف مجموعة من الخصائص المشتركة التي لا تعتمد على نوع الوظيفة مثل رقم الموظف والاسم والعنوان وتاريخ الميلاد. وهناك خصائص أخرى تعتمد على نوع الوظيفة مثل الراتب أو الأجر بالساعة أو قيمة المكافأة. وعلى ذلك فقد تم تمثيل البيانات المشتركة لجميع الموظفين من خلال كائن عام (علوى) باسم

«موظف Employee». وتم تمثيل باقي الخصائص التي تمثل الأجر من خلال ثلاثة كائنات فرعية مرتبطة بالكائن العلوي «موظف». يحتوى كل كائن فرعى على البيانات التي تميزه بالإضافة إلى مفتاح الربط «رقم الموظف». فنجد أن الكائن «بعض الوقت Part Time» يحتوى على بيان معدل الأجر بالساعة، فى حين يحتوى الكائن «دائم Full Time» على بيان الراتب السنوى، ويحتوى الكائن «استشارى Consultant» على بيان قيمة الاستشارة. ويمكن لكل كائن فرعى أن يرث (يستخدم) جميع بيانات (خصائص) الكائن العلوي «موظف»، بالإضافة إلى البيانات الخاصة به.



شكل (١٧-٣) استخدام أسلوب التعميم

٢-٢ النمذجة الموجهة بالكائنات Object-Oriented Modeling:

بعد أن استعرضنا الأسلوب التقليدى لنمذجة النظم، نقدم الآن عرضاً سريعاً للأسلوب الأكثر شيوعاً الآن فى تطوير النظم وهو أسلوب الكائنات الموجهة Object-Oriented. يعتمد هذا الأسلوب على مفهوم الكائن Object كمركب أساسى

لتطوير النظم. يتميز الكائن بتركيب خاص يقوم على تغليف كل من بيانات Data وسلوك Behavior الكائن في إطار واحد مما أدى إلى إمكانية استخدامه في نمذجة كل من بيانات وعمليات النظام في آن واحد خلال مراحل التحليل والتصميم، وكذلك يتم استخدامه في مرحلة البرمجة من خلال لغات البرمجة الموجهة بالكائنات Object-Oriented Programming. يدعم هذا الأسلوب مجموعة من الخصائص المهمة والمفيدة في بناء النظم الكبيرة والمعقدة مثل التوريث Inheritance، والتجريد Abstraction، والتغليف Encapsulation، وتعدد الأوجه Polymorphism. بالإضافة إلى الميزة الأساسية التي يتميز بها هذا الأسلوب وهي إمكانية إعادة استخدام الكائنات المصممة جيداً في عدة تطبيقات مشابهة، مما يوفر الوقت والمجهود والتكلفة أيضاً في أثناء تطوير أنظمة جديدة.

الكائن Object:

الكائن هو أى شئ حقيقى له مواصفات محددة ويؤدى مهمة معينة أى أنه له دور فعلى في الحياة العملية. ولهذا نقول إن الكائن موجود حولنا في كل مكان. الكائن يمكن أن يمثل شيئاً مادياً محسوساً أو مرئياً مثل الأشخاص أو الأشياء المادية (كتاب، جهاز، سلعة، وثيقة، ..) ويمكن أيضاً أن يمثل الكائن شيئاً غير مادي مثل مفهوم أو حدث (تسجيل طالب في دورة، حجز في رحلة طيران، حساب عميل في بنك، ميزانية، ..).

والكائن له مجموعة من الخصائص أو السمات Attributes، وله وظيفة أو سلوك Behavior. يتم وصف سلوك الكائن عن طريق مجموعة من العمليات Operations. وهي تصف طريقة تفاعل الكائن مع الكائنات الأخرى. فمثلاً يحتوى الكائن الذى يمثل شخصاً على بعض الخصائص مثل:

الاسم، النوع، تاريخ الميلاد، العنوان، ...

ويحتوى الكائن على عمليات تصف سلوكه مثل:

يقرأ ()، يكتب ()، يأكل ()،

وجدير بالذكر هنا أن العمليات الخاصة بالكائن هي فقط التي تستطيع أن تتفاعل مع بياناته (خصائصه) مباشرة، وهي بذلك تحمى الكائن من التعامل الخطر غير المتوقع من خارجه.

التجريد Abstraction:

المقصود بالتجريد هو التركيز على السمات (الخصائص والعمليات) التي نحتاج إليها الآن في أثناء عملية النمذجة وتتعلق بوجهة النظر التي يتم فحص النظام من خلالها وترك باقي السمات التي لا تتعلق بها. يختلف التجريد من مرحلة إلى أخرى في أثناء نمذجة النظام، فعلى حسب الاحتياج يكون النموذج موجزاً أو يحتوي على تفاصيل. وتكون النماذج غالباً في المستويات الأعلى موجزة، وتكون النماذج في المستويات الأدنى محتوية على تفاصيل أكثر، حتى نصل إلى أدنى مستوى من التجريد.

التوريث Inheritance:

يصف التوريث علاقات المشاركة بين بعض كائنات النظام. فعندما يشترك عدد من كائنات النظام في بعض السمات (خصائص وعمليات) وتختلف في البعض الآخر، يمكننا هنا تنظيمها في شكل صنف عام أو علوى Super Class، وأصناف فرعية Subclasses. يحتوى الصنف العلوى على السمات المشتركة بينها، وتحتوى الأصناف الفرعية على السمات الخاصة بكل كائن. ويتم تمثيل علاقة التوريث أو الوراثة بين الصنف العلوى والأصناف الفرعية والتي تعنى أن جميع الأصناف الفرعية يمكنها أن ترث سمات الصنف العلوى، بالإضافة إلى السمات الخاصة بها. وقد تم توضيح ذلك في أثناء عرض علاقة التعميم Generalization بين الكائنات من خلال الشكل (٢-١٧).

تعدد الأوجه Polymorphism:

يعنى تعدد الأوجه أنه يمكن لأصناف مختلفة أو مكونات مختلفة أن تؤدي نفسها المهمة ولكن بطريقة مختلفة. فمن المعروف في مجال البرمجيات أنه يوجد عديد من الأدوات البرمجية التي تقوم بالوظيفة نفسها ولكن بأسلوب مختلف مثل أدوات الرسم وكتابة النصوص ولفات البرمجة. ومن جانب آخر يمكن أن تحتوى أصناف مختلفة على عملية بالاسم نفسه ولكن تختلف عن بعضها في الأسلوب أو حتى في الوظيفة. فمثلاً يمكن أن يتضمن عدد من الأصناف عملية باسم «افتح» ويكون مفهومها في أحد الأصناف يعنى: «فتح جهاز»، وفي صنف آخر يعنى «فتح ملف»، وفي صنف ثالث يعنى «فتح حساب مالى»، وهكذا. ومن الواضح أنه في كل حالة سيتم تنفيذ عملية مختلفة تماماً مع أنها تأخذ نفس الاسم. وتكمن أهمية هذا المفهوم في أنه يمكننا من المحافظة على المصطلحات الطبيعية في كل مجال دون اللجوء إلى استخدام مصطلحات مصطنعة للتغلب على مشكلة تشابه الأسماء مع الاختلاف في المفهوم.

التغليف Encapsulation

يقصد بالتغليف هنا أنه يتم إخفاء تفاصيل سلوك الكائن عن الكائنات الأخرى التى تتعامل معه. فمثلاً يتكون جهاز الحاسب الآلى من عدة مكونات أو كائنات تتفاعل معاً لتحقيق أهداف استخدام الحاسب. ولكن ليس من الضروري إطلاقاً أن يكون أى مكون من الجهاز على دراية تامة بكيفية تنفيذ مكون آخر لوظيفته، ولكنه يتعامل معه ويستخدم ما يقوم به من وظيفة دون الحاجة لمعرفة كيفية تنفيذها. فمثلاً يستخدم المعالج CPU كلاً من لوحة الإدخال، والشاشة، والقرص الصلب لتنفيذ مهام معينة يحتاج إليها عند تنفيذ المهام المطلوبة منه، ولكن لا يتطلب ذلك معرفته الكيفية التى يقوم كل مكون بأداء وظيفته. نستفيد من ذلك أنه عند حدوث خلل أو عطب فى أحد المكونات، يتم صيانته أو استبداله دون التأثير فى المكونات الأخرى التى تتعامل معه. هناك ميزة أخرى لمفهوم التغليف وهو أن عملية إخفاء خصائص الكائن Information Hiding عن باقى الكائنات تؤدي إلى حماية تلك الخصائص من التعامل غير الصحيح من الخارج الذى قد يعرضها للخطر أو التدمير. ولكن الصحيح أن يتم حمايتها وذلك بقصر التعامل معها من خلال عمليات الكائن نفسه فقط ويتم تصميمها بعناية تؤدي الغرض المطلوب دون تعريض البيانات للخطر.

الرسائل Messages

تتفاعل الكائنات فيما بينها عن طريق أسلوب الرسائل، فيقوم الكائن المرسل بإرسال رسالة لكائن آخر يحدد فيها الخدمة المطلوبة. وبعد ذلك يقوم الكائن المستقبل باستقبال الرسالة، ثم يقوم بتنفيذ العملية المطلوبة. وتجدر الإشارة هنا إلى أن الرسالة يجب أن تحتوى على اسم الكائن المستقبل، ثم اسم العملية المطلوب تنفيذها مصحوبة بالمعلومات الضرورية لتنفيذ العملية.

١-٢-٢ مقدمة عن لغة UML

لقد أثبت أسلوب البرمجة المبنية على الكائنات نجاحاً مبهرًا فى تطوير التطبيقات فى الفترة من أوائل التسعينيات إلى الآن، مما أدى إلى استخدامه على نطاق واسع واعتماده أسلوباً برمجياً قياسياً لتطوير تطبيقات النظم. فكان من الطبيعى أن تواكب أساليب تحليل وتصميم النظم هذا الأسلوب فى تطوير النظم. ويتطلب ذلك استحداث طرق جديدة لنمذجة النظم تعتمد على أسلوب الكائنات الموجهة والاستفادة

من الخصائص الجديدة التي تميزه في أثناء عمليات التحليل والتصميم. وقد ظهرت عدة محاولات جادة في هذا المجال في أوائل التسعينيات من جانب كل من Booch، Rumbaugh، Jacobson. وفي منتصف التسعينيات تم تجميع هذه المحاولات وتقديم منهجية موحدة لنمذجة النظم أطلق عليها اسم «لغة النمذجة الموحدة Unified Modeling Language (UML)».

تعتبر لغة النمذجة الموحدة UML إحدى الوسائل المرئية التي تستخدم لنمذجة النظم System Modeling من خلال مجموعة من المخططات Diagrams التي تصف تركيب Structure وسلوك Behavior أى نظام معلومات. تمكن لغة UML مطوري النظم من وصف رؤيتهم لنظام برمجى معين بطريقة قياسية سهلة الفهم، وتساعدهم على تبادل وجهات النظر ومناقشتها مع مستخدمى النظام بطريقة فعالة.

تتضمن لغة UML عدة مفاهيم ورموز وإرشادات مفيدة جداً تمكن مطوري النظم من وصف النظام بدقة ومن وجهات نظر متعددة. تقوم لغة UML بتمثيل تلك المفاهيم من خلال مجموعة من المكونات والمخططات التي تمكننا من وصف النظام من منظورات Views مختلفة مثل (Rumbaugh et. Al, 2005):

- المنظور الساكن Static View أو التركيبى الذى يصف مكونات أو هيكل النظام.
- المنظور الديناميكي Dynamic View الذى يصف سلوك Behavior النظام.
- المنظور البيئي Environmental View الذى يصف البيئة المحيطة بالنظام وتفاعلها معه.
- منظور تنظيمى Organizational لتتظيم النماذج Models فى شكل حزم Packages تمكن فريق العمل من تقسيم النظم الكبيرة إلى مجموعة من النظم الفرعية مع توضيح التفاعلات بينها.

تتميز لغة UML بأنها تقوم بوصف النظام من خلال عدة نماذج (مخططات)، إذ يقوم كل نموذج بوصف النظام أو جزء منه من منظور معين، مما يؤدي إلى فهم النظام بشكل جيد، ويسهل أيضاً من عملية الاتصال والتفاهم بين أعضاء فرق التطوير بالنظام، بالإضافة إلى التفاهم والاتصال مع مجموعة مستخدمى النظام.

وعلى هذا فإنه يمكننا تعريف لغة UML بأنها لغة نمذجة متعددة الأغراض General Purpose Modeling Language. ومع أنها ليست لغة برمجة إلا أنه يمكن استخدامها من خلال إحدى أدوات هندسة البرمجيات CASE Tools لإنتاج برامج

بلغات برمجة عديدة لتنفيذ تطبيقات النظم المختلفة مثل أداة «مصمم أوراكل Oracle Designer» المقدمة من شركة أوراكل العالمية فى مجال تطوير تطبيقات نظم قواعد البيانات.

٢-٢-٢ نمذجة النظم باستخدام لغة UML،

تعتبر نمذجة النظم من السمات الرئيسية لأغلب منهجيات تطوير النظم. ولذلك فإنه لتطوير نظام معلومات بطريقة فعالة، يجب أن يتم فحص النظام من عدة منظورات، ومن ثم يتم نمذجة السمات التى تميز النظام. وكلما كان النظام كبيراً، زادت أهمية نمذجته.

يعتبر النموذج Model بمنزلة مخطط Blueprint يمثل النظام أو جزءاً منه مركزاً على وجهة نظر معينة. يمكن أن يكون النموذج موجزاً أو مفصلاً حسب الحاجة أو حسب المنظور المطلوب تمثيله. ولذلك فإن نظم المعلومات تحتاج إلى أسلوب النمذجة الذى يمكننا من وصف النظام من عدة وجهات نظر مختلفة تدعم عملية الفهم والتواصل بين المشاركين فى النظام. بالإضافة إلى أن نماذج النظام تعتبر جزءاً أساسياً من توثيق النظام الذى يساعد أى عضو جديد فى فريق العمل على فهم النظام والتعرف على خصائصه ومكوناته.

تعتبر لغة النمذجة الموحدة UML هى الأسلوب القياسى المعتمد لنمذجة نظم المعلومات. يحتوى النموذج على جزأين أساسيين هما: (Schmuller, 2004).

- رموز رسومية Notations: تمثل مكونات النماذج رسوماً والعلاقات بينها.
 - معلومات دلالية Semantic Info: معلومات تصف المعنى أو المفهوم من النموذج.
- تستخدم النماذج لعدة أغراض منها:

- ١- توصيف متطلبات النظام بطريقة دقيقة يفهمها جميع المشاركين فى النظام لدراساتها والموافقة عليها.
- ٢- مساعدة المطورين فى عرض التصميمات المقترحة كحلول للنظام واختيار الأنسب منها قبل عملية التنفيذ.
- ٣- توليد مكونات تشغيلية للنظام قابلة للاستخدام مثل هياكل الكائنات ومحتوى الإجراءات وقواعد البيانات.

٤- تنظيم المعلومات داخل النظام من خلال عدة منظورات مثل المنظور الساكن لمكونات النظام Static View، والمنظور الديناميكي لسلوك الكائنات بالنظام Dynamic View.

٥- إمكانية السيطرة على الأنظمة الكبيرة والمعقدة، وذلك باستخدام أسلوب النمذجة على عدة مستويات.

٢-٢-٣ مكونات لغة UML:

لا يوجد هناك حد فاصل بين المكونات والمفاهيم في لغة UML. ولهذا فانه من الأفضل أن نعرض مكونات لغة UML من خلال عدة منظورات Views. المنظور ببساطة هو نموذج أو أكثر يمثل النظام أو جزءاً منه من وجهة نظر أو رؤية معينة. يحتوى المنظور على مخطط Diagram أو أكثر لتمثيل المفهوم المقصود بشكل تخطيطي.

يمكننا تقسيم المنظورات إلى أربعة أنواع رئيسية هي (Fowler, 2004):

١- تركيبي Structural:

يصف مكونات النظام التركيبية والوظيفية والعلاقات بينها. يحتوى هذا التصنيف على عدة مخططات مثل مخطط الأصناف Classes Diagram، ومخطط حالات الاستخدام Use Cases Diagram، ومخطط المكونات Components Diagram.

٢- سلوك ديناميكي Dynamic Behavior:

يصف سلوك النظام من جانب التغيرات التي تحدث على مكونات النظام. يحتوى هذا النوع على مخططات مثل مخطط الحالة State Diagram، ومخطط النشاط Activity Diagram، ومخطط التفاعل Interaction Diagram.

٣- تخطيط مادي Physical Layout:

يصف هذا النوع المصادر التي يتم استخدامها لنشر النظام Deployment. يستخدم مخطط النشر Deployment Diagram لوصف هذا النوع.

٤- نموذج تنظيمي Management Model:

يصف هذا النوع الشكل التنظيمي لنماذج النظام في شكل هرمي. يستخدم مخطط الحزمة Package Diagram ومخطط المكونات Component Diagram لوصف هذا النوع. وسوف نستعرض الآن معظم المخططات التي تقدمها لغة النمذجة الموحدة UML.

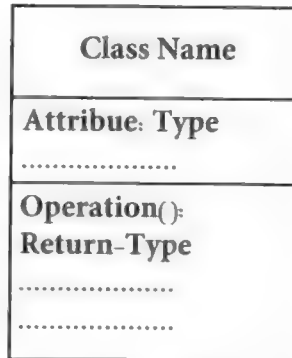
٤-٢-٢ مخطط الأصناف Class Diagram:

يعتبر مخطط الأصناف من المخططات الأساسية التي تستخدم لوصف مكونات النظام في الحالة الساكنة Static، أى أنه يصف النظام من المنظور البنائي Structural View أو المنظور الساكن Static View. يصف مخطط الأصناف أنواع الكائنات الموجودة بالنظام والعلاقات بينها في حالة السكون. يصف المخطط أيضاً خصائص وعمليات كل صنف في النظام، بالإضافة إلى القيود Constraints التي تحكم اتصال الكائنات بعضها ببعض. يتم توصيف مجموعة الكائنات المشتركة في التركيب Structure والسلوك Behavior في شكل صنف Class يحتوى على خصائص Attributes الكائنات والعمليات Operations التي تصف سلوكها. يعتبر الصنف Class هو المكون الأساسى لمخطط الأصناف، ويتم تمثيله من خلال صندوق يتكون من ثلاثة أجزاء هي (شكل (٢-١٨):

- اسم الصنف Class-Name.

- الخصائص Attributes.

- العمليات Operations.



شكل (٢-١٨) الشكل العام للصنف Class

الخصائص Attributes،

تعتبر الخاصية هي إحدى السمات التي تميز صنف الكائن وتصف مجموعة القيم التي يمكن أن يأخذها الكائن. يمكن أن يحتوى الصنف على عديد من الخصائص.

الشكل العام للخاصية فى لغة UML هو:

Visibility name: type multiplicity = default {property string}

حيث تحدد visibility مدى استخدام الخاصية، وهى إما عام Public ويرمز لها بالرمز (+)، أو خاص Private ويرمز لها بالرمز (-)، وسيتم شرحها فيما بعد.

وتحدد multiplicity عدد الكائنات التي يمكن أن تأخذ الخاصية (صفر أو واحد أو أكثر).

وتمكننا default من إضافة قيمة افتراضية للخاصية.

وتمكننا property string من إضافة أى معلومة أخرى للخاصية مثل read only لتوضيح عدم إمكانية التعديل فى قيمة الخاصية. يوضح شكل (٢-١٩) مثلاً لخصائص الصنف order.

order
+ dateReceived: Date (0.001) + isPrepaid: Boolean (1) + rintItems: orderLine (*)
+calculate Total: double

شكل (٢-١٩) خصائص وعمليات الصنف order

العمليات Operations،

تمثل العمليات مجموعة الأفعال Actions التي يمكن أن يقوم بها صنف الكائن. وكما ذكرنا من قبل، فإن العمليات تمثل واجهة الصنف، حيث يتم التعامل الخارجى مع

الكائن من خلالها فقط، وهي التي تقوم بدورها بالتعامل مع خصائص الكائن. هناك نوعان أساسيان من العمليات هما:

١- عمليات تقوم بتحديد قيم خصائص الصنف مع إمكانية التعديل فيها، أي أنها تغير من حالة الكائن، وتسمى Setting Methods أو Modifiers.

٢- عمليات تقوم بالاستعلام عن قيم خصائص الصنف، ولا تقوم بتغييرها، أي أنها لا تغير من حالة الكائن، وتسمى Getting Methods أو Query.

الشكل العام للعملية في لغة UML هو:

Visibility name(parameter-list): return-type {property string}

تمثل return-type نوع القيم المسترجعة من العملية في حالة Query.

العلاقات Relationships

تحتوى لغة UML على مجموعة من العلاقات المختلفة التي تربط بين مكونات اللغة. يعرض شكل (٢-٢٠) مجموعة العلاقات ووصفاً مختصراً لها، بالإضافة إلى الرموز المستخدمة لتمثيل كل علاقة في لغة UML.

العلاقة	الوصف	الرمز
الارتباط Association	تصف عملية الربط بين كائنات الأصناف المختلفة Objects.	
التعميم Generalization	تصف العلاقة بين الأصناف العلوية Super-classes والأصناف الفرعية Sub-classes في أثناء عملية التوريث Inheritance.	

التحقق Realization	تصف العلاقة بين صنف الواجهة Interface والأصناف التي تحققه أو تطبق مواصفاته المملوكة Implementation.	←-----
الاعتمادية Dependency	تصف علاقة الاعتمادية الموجودة بين صنفين يكون لأحدهما تأثير في الآخر.	←-----

شكل (٢-٢٠) أنواع العلاقات في لغة UML

وفيما يلي وصف موجز لكل نوع من أنواع العلاقات بين الأصناف في لغة UML.

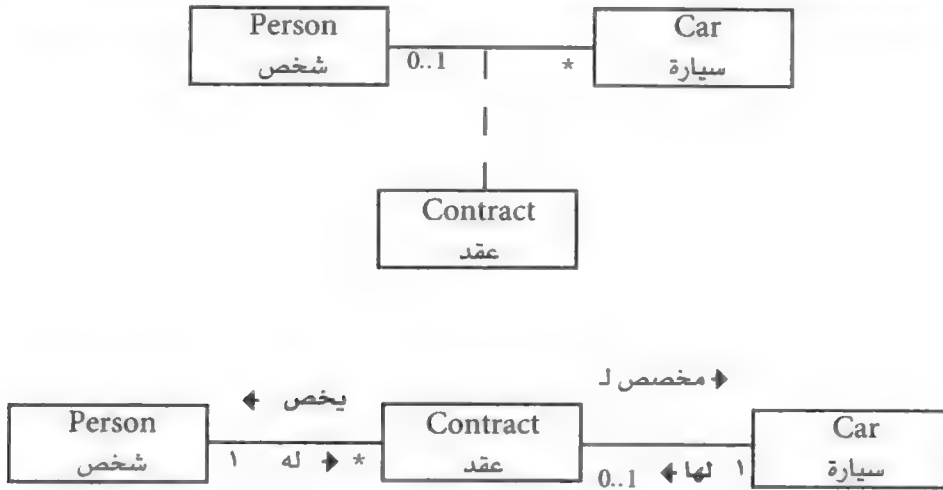
الارتباطات Associations

يتم توصيف الارتباط المنطقي بين أصناف الكائنات باستخدام خط توصيل بين الأصناف يحدد نوعية الارتباط. ويمكن إضافة مقطع نصي لتوضيح نوع الارتباط كما هو موضح بشكل (٢-٢١).



شكل (٢-٢١) الارتباط بين أصناف الكائنات

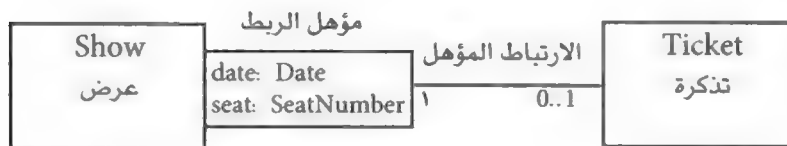
أحياناً تحتوى عملية الارتباط نفسها على بعض الخصائص والعمليات، وهنا يجب تمثيل عملية الارتباط من خلال صنف خاص يسمى صنف الارتباط Associative Class، ويتم ربطه بالأصناف الأخرى داخل النظام. كمثال على ذلك فإنه يمكننا تمثيل عملية الارتباط بين الشخص والسيارة في شكل صنف ارتباط يسمى «عقد» يصف عملية الارتباط بين الصنفين من خلال عملية تحرير عقد يحتوى على عدة خصائص منها تاريخ البيع والسعر وطريقة الدفع. يوضح شكل (٢-٢٢) كيفية تمثيل صنف الارتباط «عقد» بطريقتين مختلفتين في لغة UML.



شكل (٢-٢٢) صنف الارتباط

الارتباطات المؤهلة، Qualified Associations

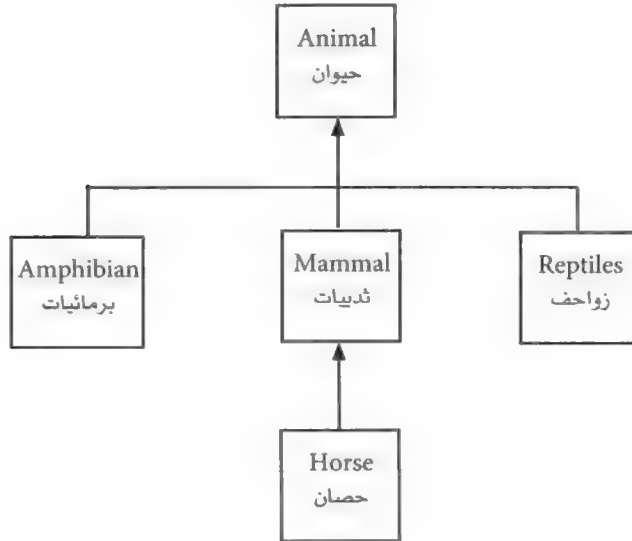
عندما تكون قيمة أحد الخصائص للصنف هي المؤهلة لاختيار الكائنات المقابلة لها في صنف آخر مرتبطة به، فإن هذه الخاصية تسمى مؤهل الربط Qualifier. وعلى هذا الأساس يتم اعتماد قيم هذه الخاصية لاختيار كائنات معينة من مجموعة الكائنات المرتبطة التي تحقق عملية الارتباط بين الصنفين. تستخدم لغة UML رمز المستطيل ملاصقاً للصنف الذي يحتوي على الخاصية المؤهلة. يوضح شكل (٢-٢٣) مثالاً على استخدام الارتباط المؤهل حيث تم اختيار مؤهل الربط المكون من خصائص «رقم المقعد Seat Number» و «التاريخ Date» لتحقيق عملية الربط بين الصنف «عرض» والصنف «تذكرة» مما يعنى أن كل تذكرة تكون مخصصة لمقعد محدد في تاريخ محدد لأحد العروض.



شكل (٢-٢٣) الارتباط المؤهل

التعميم Generalization

تعتبر خاصية التوريث من المفاهيم الأساسية لأسلوب الكائنات الموجهة. وتكمن أهميتها في أنها موجودة بكثرة في معظم الأنظمة التي نتعامل معها في حياتنا اليومية. يتم تمثيل الخصائص والعمليات المشتركة في شكل صنف عام general class أو صنف علوى super class. ويتم تمثيل الخصائص والعمليات التي تخص كل نوع في شكل أصناف فرعية sub classes ترتبط بالصنف العلوى من خلال علاقة التعميم. يتم تمثيل علاقة التعميم باستخدام سهم على شكل مثلث يبدأ من الأصناف الفرعية ويتجه نحو الصنف العلوى. تسمح هذه العلاقة للأصناف الفرعية أن تستخدم جميع الخصائص والعمليات الموجودة بالصنف العلوى مع إمكانية تغييرها، وذلك بالإضافة إلى الخصائص والعمليات الموجودة بكل صنف فرعى. أحياناً يطلق على الصنف العلوى اسم الصنف الأب parent class والصنف الفرعى اسم الصنف الابن child class. يمكننا تمثيل تلك العلاقة على عدة مستويات، فيمكن للصنف الابن أن يكون أباً لصنف أو أصناف أخرى فرعية. يوضح شكل (٢-٢٤) مثالاً لتمثيل خاصية التعميم في عدة مستويات. نلاحظ في هذا المثال أن الصنف «حصان horse» هو نوع من الصنف «ثدييات mammal» وهو يعتبر أيضاً نوعاً من الصنف «حيوان animal».



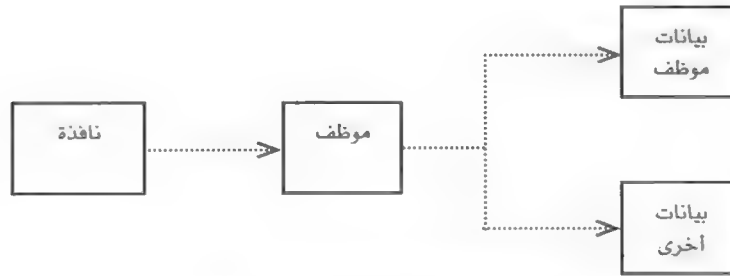
شكل (٢-٢٤) مثال لخاصية التعميم

الاعتمادية Dependency

تعتبر الاعتمادية نوعاً من العلاقات بين الأصناف التى يقوم فيها أحد الأصناف «عميل client» باستخدام صنف آخر «مزود supplier»، ويعنى هذا أن أى تغيير فى الصنف supplier يؤدي إلى تغيير فى الصنف client، ولذلك فهو يعتبر معتمداً عليه. توجد الاعتمادية بين الأصناف لعدة أسباب منها:

- أن يرسل صنف رسالة إلى صنف آخر لاستخدام إحدى وظائفه.
- أن يكون صنف ما عبارة عن جزء من بيانات صنف آخر.
- أن يكون أحد الأصناف عبارة عن متغير ممرر passing parameter لإحدى عمليات صنف آخر.
- أن يكون أحد الأصناف معرفاً كمتغير محلى داخل إحدى عمليات صنف آخر.
- أن يكون أحد الأصناف معرفاً كمتغير عام Global فيصبح متاحاً لكل العمليات.

يتم تمثيل الاعتمادية فى لغة UML باستخدام خط متقطع ينتهى بسهم موجه ناحية الصنف supplier، كما هو موضح بالشكل (٢-٢٥). يتضح من الشكل أن الصنف «نافذة» يعتمد على الصنف «موظف» الذى بدوره يعتمد على الأصناف «بيانات موظف» و«بيانات أخرى» وهى تزوده ببيانات الموظف التى يتم عرضها من خلال صنف النافذة.

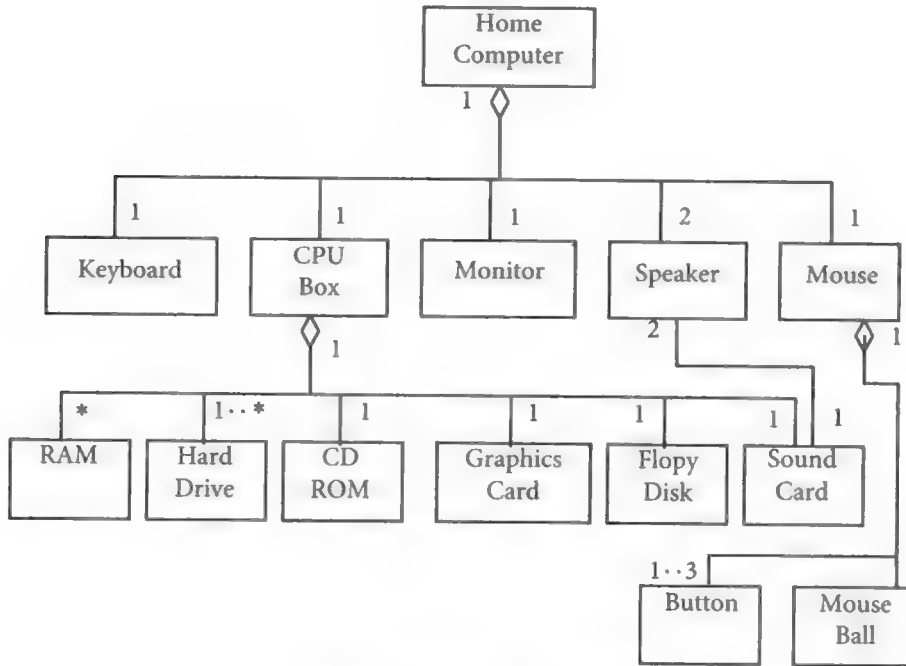


شكل (٢-٢٥) مثال للاعتمادية

التجميع Aggregation

أحياناً يحتوى النظام على صنف عام (مجمع) وهو عبارة عن مجموعة من الأصناف الأخرى التى تم تجميعها فى شكل الصنف المجمع. وهنا نجد أن هناك علاقة تجمع

بين الصنف المجمع والأصناف المكونة له وهي تسمى علاقة «جزء-كل» "part-whole" أو علاقة التجميع Aggregation. فمثلاً يمكننا وصف العلاقة بين جهاز حاسب وأجزائه المختلفة في شكل علاقة تجميع. يوضح شكل (٢-٢٦) تلك العلاقة حيث يعتبر الصنف "Home Computer" هو الصنف المجمع "whole"، وهو يتكون من عدة أجزاء، كل جزء في شكل صنف "part" مثل "keyboard"، "monitor"، "mouse"، "CPU-Box". ونجد أيضاً أن أحد هذه الأجزاء "CPU-Box" يتكون من عدة أجزاء، ولهذا يعتبر صنفاً مجمعاً "whole" لأصناف أخرى مكونة له في شكل أصناف "part" مثل "CR-ROM"، "RAM"، "Hard-Drive". وكذلك نجد أن الصنف "mouse" هو بمنزلة صنف مجمع لكل من "Mouse-Ball"، "Button". يتم تمثيل علاقة التجميع في لغة UML في شكل خطوط تصل الأجزاء بالمجمع لها وتنتهي بشكل «معين مفرغ» يتجه نحو الصنف المجمع.



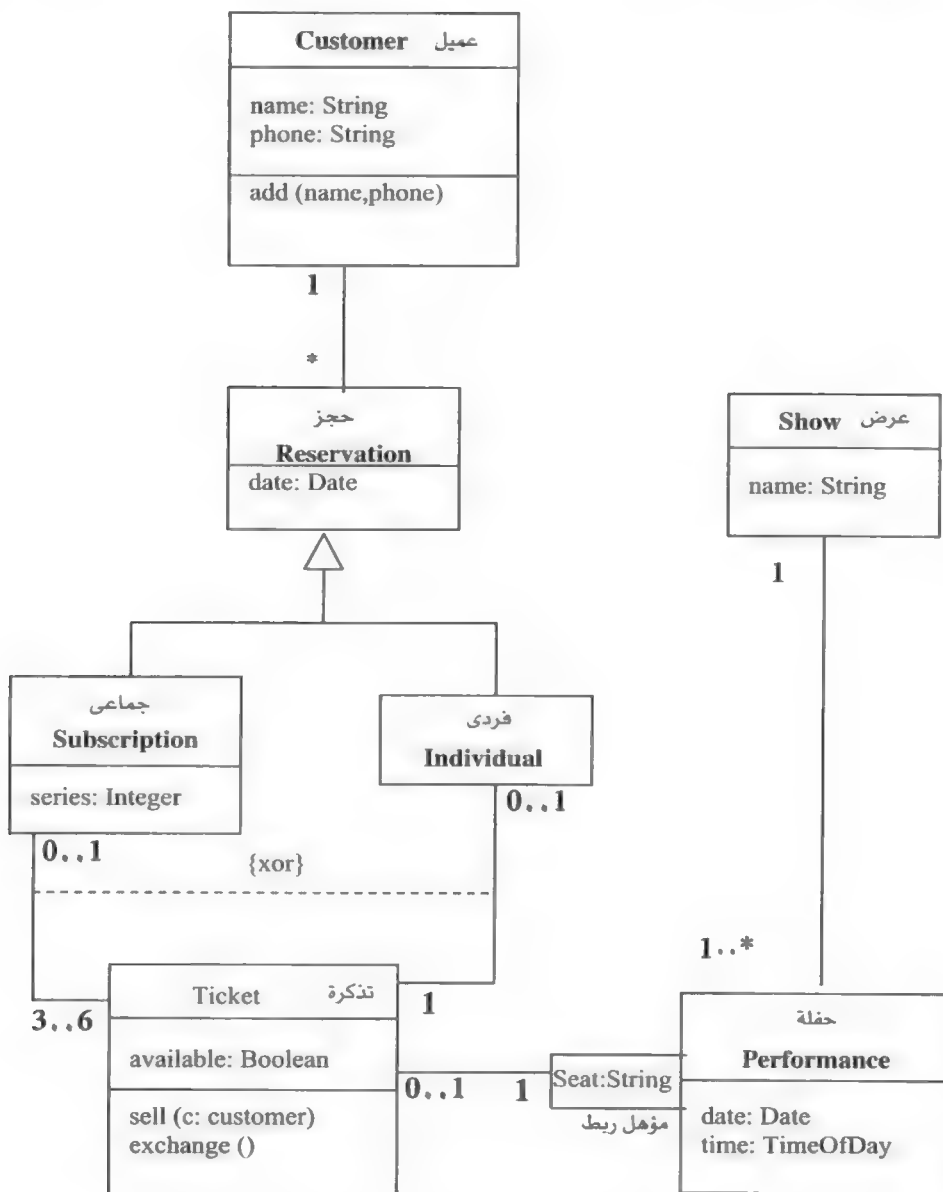
شكل (٢-٢٦) علاقة التجميع

مثال لاستخدام مخطط الأصناف:

يوضح شكل (٢-٢٧) مثلاً لاستخدام مخطط الأصناف لوصف الشكل البنائى لنظام حجز وبيع التذاكر فى إحدى دور العرض الترفيهية. يتميز هذا النظام باحتوائه على معظم أنواع العلاقات بين الأصناف التى تم ذكرها آنفاً. يحتوى المخطط على مجموعة الأصناف الأساسية التى تمثل عملية الحجز مثل «عميل Customer» و«حجز Reservation» و«تذكرة Ticket» و«حفلة Performance» و«عرض Show». تحتوى عملية الحجز على نوعين: الأول هو الحجز الفردى لشخص واحد، والثانى هو الحجز الجماعى لمجموعة من الأشخاص لا تقل عن ثلاثة ولا تزيد على ستة أشخاص. يتم تنفيذ العرض الترفيهى فى أكثر من حفلة فى اليوم أو فى الأسبوع. يبدأ العميل بتحديد عملية الحجز (فردى/جماعى) ويمكنه أن يقوم بإجراء حجز واحد أو أكثر. يتم تمثيل عملية الحجز باستخدام أسلوب علاقة التعميم من خلال صنف عام «حجز» وصنفين فرعيين له هما «فردى»، «جماعى» كما هو موضح بالشكل. وبعد ذلك يتم استخراج التذاكر للعميل، وذلك بإصدار تذكرة للحجز الفردى وعدد من التذاكر يناسب عدد الأشخاص فى الحجز الجماعى. تصدر التذكرة للعميل فى مقعد معين فى حفلة معينة (بتاريخ ووقت محددين) لعرض معين. ويتم تمثيل ذلك من خلال علاقة الربط المؤهل بين صنف «تذكرة» وصنف «حفلة» باستخدام خاصية «المقعد Seat» باعتبارها مؤهلاً لعملية الربط.

٢-٢-٥ مخطط حالات الاستخدام Use Cases Diagram:

يستخدم مخطط حالات الاستخدام لوصف المتطلبات الوظيفية للنظام. يتم تحديد تلك المتطلبات فى أثناء مرحلة التحليل من خلال مجموعة التفاعلات Interactions التى تحدث فعلياً بين مستخدمى النظام والنظام نفسه. فكل مستخدم له طريقته فى التعامل مع النظام من خلال مجموعة من الإجراءات فى شكل سيناريو معين Scenario. والسيناريو هو عبارة عن مجموعة الخطوات الإجرائية التى تصف تفاعل المستخدم مع النظام لتنفيذ وظيفة محددة. يتم وصف أى إجراء وظيفى يتفاعل به المستخدم مع النظام (سيناريو) من خلال وحدة وظيفية تسمى «حالة استخدام Use Case». وبالمطابق فإن تعامل المستخدمين مع النظام يمكن حصره فى شكل مجموعة من السيناريوهات أو حالات الاستخدام Use Cases.



شكل (٢٧-٢) مثال لمخطط الأصناف

يستخدم مخطط حالات الاستخدام لوصف تفاعل مجموعة المستخدمين مع النظام من خلال مجموعة من حالات الاستخدام. تكمن أهمية هذا المخطط أثناء عملية التحليل للنظام، حيث يتم استخدامه لوصف متطلبات النظام الوظيفية من وجهة نظر مستخدمى النظام.

الفاعل Actor،

الفاعل هو أى مستخدم للنظام يقوم بدور فعال فى أثناء التعامل معه. ويمكن أن يكون الفاعل شخصاً Person أو جهازاً Device أو أى نظام فرعى Sub-system آخر يتفاعل مع النظام الحالى. والفاعل من النوع (شخص) يمكن أن يكون عميلاً Customer أو مندوب مبيعات Salesperson أو مديراً Manager أو غيره. الفاعل من النوع (جهاز) يمكن أن يكون قارئاً آلياً لأكواد الأصفان Bar-code أو كاميرا أو خطاً هاتفياً أو غيره. الفاعل من النوع (نظام فرعى) يمكن أن يكون نظاماً آخر يتكامل مع النظام الحالى وأنظمة أخرى لتكوين نظام كبير أو نظام آخر خارجى يتطلب التعامل مع النظام الحالى.

نموذج حالة استخدام Use Case Model،

يبدأ الفاعل Actor عادة فى التعامل مع حالة الاستخدام Use Case بطريقة ما مثل إدخال بيان أو إرسال رسالة، ويمكنه أيضاً أن يستقبل مخرجات من حالة الاستخدام. يعتبر مخطط حالات الاستخدام بمنزلة شكل رسومى يوضح جميع حالات استخدام النظام من قبل مستخدمى النظام. وهو يشابه مخطط السياق Context Diagram المستخدم فى طريقة التحليل الهيكلى. فهو يصف حدود النظام وتفاعله مع المستخدمين من الخارج، وهو يحتوى على:

- الفاعلون Actors.

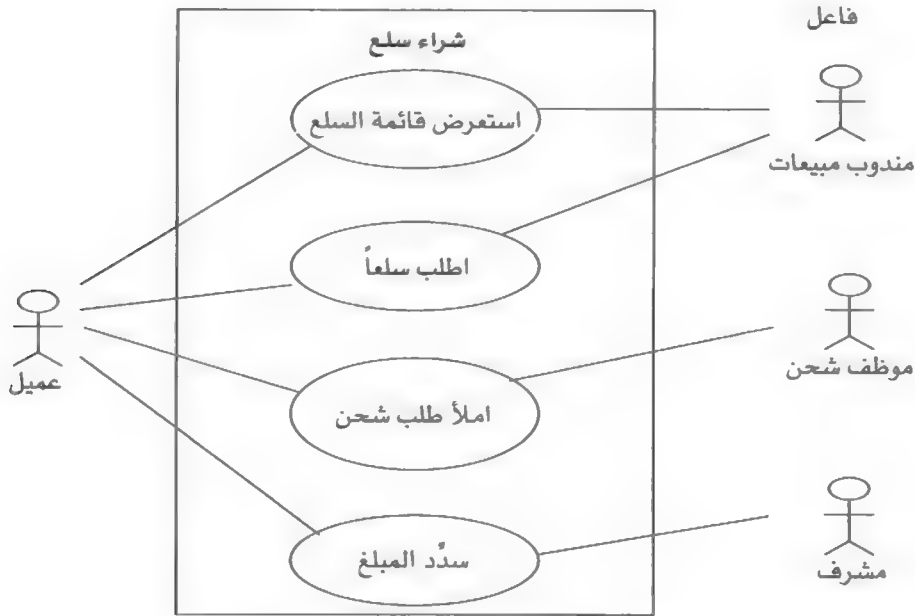
- حالات الاستخدام Use Cases.

- العلاقات بين حالات الاستخدام والفاعلين Relationships.

يمكننا استخدام مخطط حالات الاستخدام لوصف نظام كامل أو وصف نظام فرعى (جزء من نظام أكبر) أو لوصف أجزاء داخلية صغيرة مثل حالات استخدام صنف معين. جدير بالذكر هنا أنه إذا كان محتوى حالة الاستخدام كبيراً بحيث يحتوى على تفاصيل عديدة، فيمكننا أن نجزئها إلى مجموعة من حالات الاستخدام البسيطة.

يوضح شكل (٢-٢٨) مثالاً لاستخدام مخطط حالات الاستخدام لوصف المتطلبات الوظيفية لعملية شراء السلع. يحتوى المخطط على أربعة فاعلين «عميل» و«مندوب مبيعات» و«موظف شحن» و«مشرف». يقوم العميل باستعراض قائمة السلع بالاشتراك مع مندوب المبيعات لاختيار السلع المطلوب شراؤها من خلال حالة الاستخدام «استعرض سلعاً». بعد ذلك يقوم النظام بعرض بيانات أسعار السلع المختارة وتكاليف شحنها، ومن ثم يقوم العميل بتعديل القائمة (بالحذف أو الإضافة) ثم يقوم بتأكيد عملية الطلب من خلال حالة الاستخدام «اطلب سلعاً». بعد ذلك يقوم موظف الشحن بالاشتراك مع العميل بملء بيانات طلب الشحن من خلال حالة الاستخدام «املاً طلب شحن». ثم يقوم العميل بالدفع الفوري أو ببيانات الدفع بشيك مصرفى أو ببيانات التسديد الإلكتروني باستخدام بطاقات الائتمان.

يتم الاستفادة أيضاً من مخطط حالات الاستخدام فى تحديد الأصناف Classes المستخدمة بالنظام، وتحديد العمليات لكل صنف (باستخدام محتوى حالة الاستخدام).



شكل (٢-٢٨) مخطط حالات الاستخدام

محتوى حالة الاستخدام Use Case Content:

كما ذكرنا من قبل فإن حالة الاستخدام تمثل أحد سيناريوهات التفاعل بين المستخدم والنظام، وأن السيناريو يتكون من مجموعة من الخطوات الإجرائية لتنفيذ مهمة أو وظيفة معينة. وكما لاحظنا فإن تلك الخطوات لا تظهر فى مخطط حالات الاستخدام، وإنما تظهر الحالات فقط وتفاعلها مع المستخدمين. لأن مخططات حالات الاستخدام تعتبر جزءاً مهماً من وثائق تحليل وتصميم النظام، ويتم الرجوع إليها دائماً من قبل فريق التطوير بالنظام لاستخراج مكونات مخطط الأصناف وتحديد الخصائص والعمليات المطلوبة وكذلك أثناء تحديد حالات الاختبار للوحدات الوظيفية، فإنه يتحتم علينا إعداد محتوى جميع حالات الاستخدام الموجودة فى المخططات وتضمينها مع وثائق النظام لتتم عملية الفهم لفريق التطوير. لم تقرض لغة UML شكلاً محدداً لوصف محتوى حالات الاستخدام ولكنه من الأفضل أن تحتوى على ما يلى:

- اسم حالة الاستخدام، ووصف مختصر لها.
 - اسم الفاعل الأساسى الذى يبدأ تنفيذ حالة الاستخدام.
 - مجموعة الافتراضات Assumptions (إن وجدت).
 - الشروط المسبقة لحالة الاستخدام Preconditions.
 - خطوات تنفيذ حالة الاستخدام Steps.
 - الشروط اللاحقة بعد انتهاء السيناريو Postconditions.
 - اسم الفاعل المستفيد من مخرجات حالة الاستخدام.
 - استخدام الامتدادات Extensions فى حالة وجود شرط يؤدي إلى إجراءات مختلفة.
- يوضح شكل (٢-٢٩) محتوى حالة الاستخدام «شراء سلع» الموضحة بالشكل (٢-٢٨).



شكل (٢٩-٢) مثال لمحتوى حالة استخدام

العلاقات بين حالات الاستخدام Relationships Among Use Cases:

تتفاعل حالات الاستخدام داخل المخطط بعضها مع بعض وكذلك مع الفاعلين بعدة أشكال. يتم تمثيل أشكال التفاعلات بين حالات الاستخدام ومع الفاعلين من خلال مجموعة من العلاقات مثل «الارتباط Association» و «التعميم Generalization» و «التضمين Inclusion» و «الامتداد Extension» كما هو موضح بالشكل (٢٩-٢). تنشأ علاقة الارتباط طبيعياً بين الفاعل وحالات الاستخدام للنظام. أما باقى العلاقات (وهى بين حالات الاستخدام فقط) فتكون نتيجة لعملية إعادة تنظيم مكونات حالات الاستخدام الأولية لتلائم خصائص ومميزات أسلوب الكائنات الموجهة الذى يتم اتباعه فى أثناء عمليات تحليل ونمذجة متطلبات النظام. فإذا كانت هناك مجموعة من

حالات استخدام النظام تشترك في بعض المتطلبات، فإنه يمكننا أن نستخدم علاقة التعميم لفصل الجزء المشترك في شكل حالة استخدام عامة، وترتبط بعلاقة التعميم مع حالات الاستخدام الفرعية والمكملة لها حسب كل متطلب. وبالمثل يمكننا استخدام علاقة التضمين بين حالة استخدام معينة وحالات استخدام أخرى تكون معدة مسبقاً لتكون ضمن مكونات حالة الاستخدام الحالية، مما يدعم أسلوب إعادة الاستخدام Reusability الذي يتميز به أسلوب الكائنات الموجهة. وكذلك يمكننا استخدام علاقة الامتداد بين حالة استخدام معينة وحالات استخدام أخرى جديدة تنشأ في أثناء عمليات التحديث والصيانة للنظام، وذلك بدون التعديل في مكونات حالة الاستخدام الحالية.

العلاقة	الوصف	الرمز
الارتباط Association	تصف عملية الربط بين حالات الاستخدام والفاعلين Actors.	_____
التعميم Generalization	تصف العلاقة بين حالة استخدام عامة Parent وحالات استخدام فرعية Child ترث خصائص الحالة العامة وتضيف إليها Inheritance.	←
التضمين Inclusion	إمكانية استخدام حالات استخدام أخرى من داخل حالة استخدام معينة.	«Include» ←-----
الامتداد Extension	إمكانية إضافة حالات استخدام جديدة إلى حالة استخدام معينة.	«Extend» ←-----

شكل (٢-٣٠) العلاقات بين حالات الاستخدام

وفيما يلي وصف مختصر لكل نوع من أنواع العلاقات في مخطط حالات الاستخدام.

التضمين Inclusion،

يتطلب أحياناً أن تتضمن إحدى حالات الاستخدام سلوك behavior حالات استخدام أخرى كجزء من سلوكها. وهنا يتم ربط حالة الاستخدام بالحالات المتضمنة للسلوك

المطلوب من خلال علاقة تسمى «تضمين Inclusion». يتم تمثيل تلك العلاقة في لغة UML مثل علاقة الاعتمادية بين الأصناف، وهى عبارة عن خط متقطع ينتهى بسهم يشير ناحية الحالات المتضمنة، ويكتب بجواره كلمة «Include»، كما فى الشكل (٢-٣١).

الامتداد Extension

يتطلب فى أحيان أخرى أن تكون حالة استخدام معينة هى امتداد إضافى Incremental Extension لحالة استخدام أساسية Base Use Case. ومن الممكن أن تكون هناك أكثر من حالة استخدام امتداداً لحالة الاستخدام الأساسية. يتم ربط حالات الاستخدام (الامتداد) بحالة الاستخدام (الأساسية) من خلال علاقة الامتداد Extension. يتم تمثيل تلك العلاقة فى لغة UML مثل علاقة التضمين، ولكن يكون اتجاه السهم ناحية حالة الاستخدام الأساسية، ويكتب جوارها كلمة «Extend»، كما فى شكل (٢-٣١).

التعميم Generalization

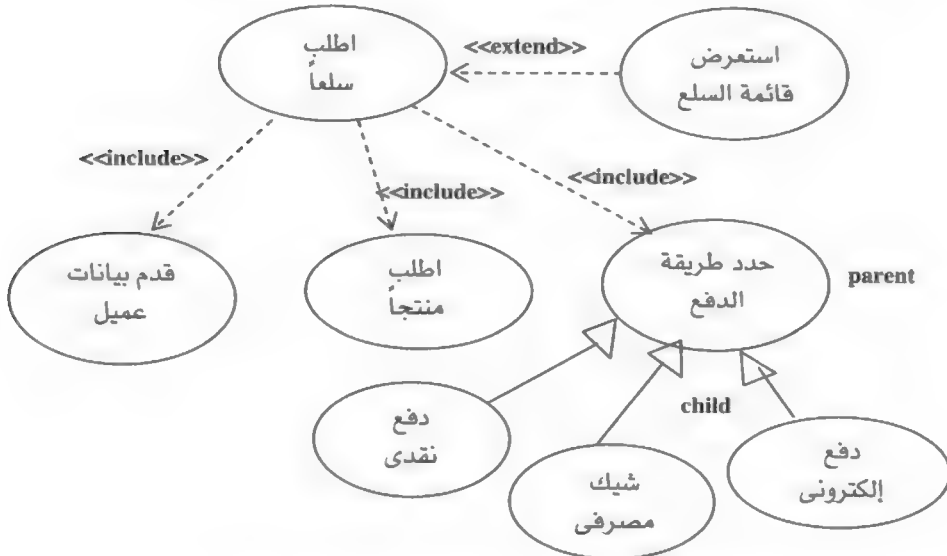
يمكننا تطبيق مفهوم التوريث Inheritance بين حالات الاستخدام أيضاً. فمن الممكن تصميم حالات الاستخدام المشتركة فى بعض السلوك فى شكل حالة استخدام عامة أو أب Parent Use Case ومجموعة من حالات الاستخدام الفرعية أو الأبناء Child use Cases. وهنا يمكن للحالات الفرعية أن ترث سلوك الحالة العامة وتضيف إليها سلوكها الخاص بها. يتم تمثيل تلك العلاقة كما هى تماماً بين الأصناف وتسمى أيضاً علاقة التعميم Generalization بين حالات الاستخدام. ويشير المثلث (رأس العلاقة) إلى الحالة العامة، كما فى شكل (٢-٣١).

التجميع Grouping

عندما يكون عدد حالات الاستخدام كبيراً، فمن الأفضل تنظيم المخطط فى شكل مجموعات. وكل مجموعة تحتوى على عدد من حالات الاستخدام المتفاعلة لأداء غرض معين. يحدث ذلك غالباً عندما يكون النظام مكوناً من مجموعة من الأنظمة الفرعية أو يمكن تجزئته إلى مجموعة من المكونات الوظيفية. ونحتاج هذه الخاصية دائماً فى أثناء مرحلة توصيف متطلبات النظام، حيث يتم وصف كل متطلب فى شكل

مخطط حالات استخدام منفصل، ثم يتم تجميع تلك المخططات في شكل مخطط مجمع للنظام.

يوضح شكل (٢-٣١) أسلوباً آخر لوصف مخطط حالات الاستخدام لعملية «شراء سلع» المقدمة في الشكل (٢-٢٨) وذلك بعد إعادة تنظيم حالات الاستخدام لتلائم أسلوب إعادة الاستخدام لبعض المتطلبات الوظيفية للنظام. فقد تم فصل عمليات طلب المنتج وتقديم بيانات العميل من حالة الاستخدام «اطلب سلعاً» ووضعها في شكل حالات استخدام منفصلة «اطلب منتجاً» و «قدم بيانات عميل»، وتم ربطهما بحالة الاستخدام الأصلية «اطلب سلعاً» من خلال علاقة التضمين، وذلك لإتاحة إمكانية استخدامهما من قبل حالات استخدام أخرى داخل النظام أو خارجه. وكذلك تم إعادة تنظيم عملية الدفع من خلال حالة استخدام عامة «حدد طريقة الدفع» ويرتبط بها ثلاث حالات فرعية لثلاثة أنواع لتسديد فاتورة شحن السلع وهي «دفع نقدي» و «دفع إلكتروني» و «شيك مصرفي». وقد تم ربط حالة الاستخدام العامة «حدد طريقة الدفع» بحالة الاستخدام «اطلب سلعاً» من خلال علاقة التضمين، إذ إنها تعتبر جزءاً من عملية طلب السلع. وقد تم أيضاً إضافة حالة الاستخدام «استعرض قائمة السلع» إلى حالة الاستخدام «اطلب سلعاً» باستخدام علاقة الامتداد.



شكل (٢-٣١) العلاقات في مخطط حالات الاستخدام

٢-٢-٦ مخطط التتابع Sequence Diagram

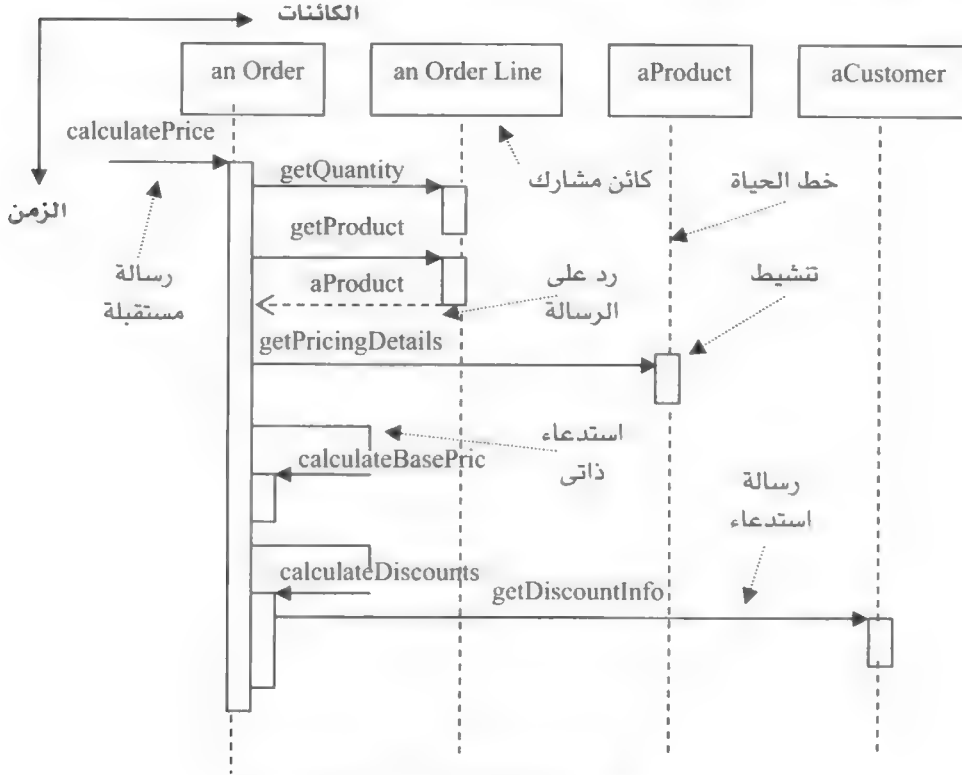
يمكننا التعرف على سلوك النظام من خلال التفاعل الذي يتم بين كائنات النظام. هناك طريقتان لوصف هذا التفاعل. يتم التركيز في الطريقة الأولى على سلوك كائن واحد، وهو ما سيتم تقديمه من خلال مخطط الحالة. وتتضمن الطريقة الثانية مجموعة من الكائنات المتفاعلة التي تمثل وجهة نظر مفصلة لسلوك النظام أو جزء منه من خلال سيناريو معين. تحدث التفاعلات بين الكائنات في شكل خطوات متتالية، وكل خطوة تستغرق فترة زمنية محددة. وعلى هذا، فإنه لا بد من تحديد السيناريو أولاً، ثم نبدأ بتمثيل التفاعلات بين الكائنات طبقاً لهذا السيناريو. تستخدم لغة UML مخطط التتابع Sequence Diagram لتمثيل هذا النوع من التفاعلات. يتم التفاعل بين الكائنات من خلال الرسائل الممررة بينها. يتم رسم خط رأسى متقطع من الكائن إلى أسفل المخطط يسمى «خط الحياة للكائن Object's Life Line». نرسم مستطيلاً رأسياً ضيقاً حول خط الحياة للكائن يسمى «مستطيل التنشيط Activation»، وهو يمثل تنفيذ عملية يقوم بها الكائن. يمثل طول مستطيل التنشيط الفترة الزمنية المطلوبة لتنفيذ العملية. يتم تمثيل الرسالة من كائن إلى آخر بخط له سهم يبدأ من خط الحياة للكائن المرسل إلى خط الحياة للكائن المستقبل للرسالة.

يتم تمثيل الوقت في المخطط في الاتجاه الرأسى ويبدأ من أعلى متجهاً إلى أسفل. يتم تمثيل الرسائل حسب الأسبقية، الرسائل الموجودة أعلى المخطط تبدأ قبل التي تليها لأسفل. وعلى هذا، فإن مخطط التتابع يعتبر ثنائى الأبعاد. تمثل الكائنات الاتجاه الأول (الأفقى) من اليسار إلى اليمين، ويمثل الوقت الاتجاه الثانى (الرأسى) ويتجه من أعلى إلى أسفل.

مثال لمخطط التتابع:

يوضح شكل (٢-٢٢) مثالاً لاستخدام مخطط التتابع، حيث يصف سيناريو تنفيذ إحدى حالات الاستخدام فى نظام طلبات العملاء الخاصة بحساب تكلفة أحد طلبات العملاء. يقوم الكائن an Order باستدعاء الكائن an Order Line لمعرفة السلع المطلوبة وكميتها من خلال الرسالتين getProduct ، getQuantity. وبعد ذلك يقوم الكائن an Order باستدعاء الكائن aProduct لمعرفة سعر كل سلعة موجودة فى الكائن an Order Line من خلال الرسالة getPricingDetails، ثم يقوم بحساب إجمالى السعر لكل سلعة من خلال استدعاء الطريقة calculateBasePrice الموجودة

داخل الكائن نفسه an Order. وبعد ذلك يقوم الكائن an Order بالحصول على نسبة الخصم Discount من الكائن aCustomer من خلال الرسالة getDiscountInfo. نقوم بعد ذلك بحساب التكلفة النهائية لطلب العميل. نلاحظ في هذا المثال أنه لم يتم تمثيل الرسائل المسترجعة لجميع الرسائل المرسلة في المخطط، ولكن تم تمثيلها في حالة واحدة فقط، وذلك لتجنب كثرة الخطوط وتبقى الفكرة منطقية وهي أن جميع الرسائل من النوع Call تحتاج إلى رسائل مسترجعة لها.



شكل (٢-٣٢) مثال لمخطط التتابع

٧-٢-٢ مخطط الحالة State Diagram

تمر كائنات النظام بمجموعة من التغيرات في أثناء تفاعلها مع المستخدمين أو أنظمة أخرى. يمكن لأي كائن أن يتفاعل مع كائنات أخرى في النظام عن طريق

الرسائل المتبادلة بينهم، مما يؤدي إلى نوع من التغيير في حالة الكائنات. وأحياناً يحدث تغيير في حالة الكائن مع مرور الوقت وبدون تأثير من كائن آخر فيه. لا بد من إيجاد وسيلة مناسبة لنمذجة تلك التغيرات التي تطرأ على الكائنات. تستخدم لغة UML مخطط الحالة باعتباره أسلوباً فعالاً لوصف السلوك الديناميكي لكائنات النظام وذلك بتمثيل التغيرات التي تحدث لكائنات النظام مع الوقت أو نتيجة أحداث معينة. فبعض الكائنات تتغير حالتها مع الوقت مثل أى عملية زمنية (مرتبطة بوقت محدد). وبعض الكائنات تتغير حالتها نتيجة حدث معين مثل بدء تشغيل أو إيقاف عملية أو إجراء معين.

يتكون مخطط الحالة من مجموعة من الحالات States والانتقالات بينها Transitions. وهو يصف رد فعل كائن معين للأحداث التي تواجهه. وبأسلوب آخر، فإن مخطط الحالة هو نموذج يصف جميع أشكال (حالات) كائن لصنف معين. فعند تعرض الكائن لحدث معين Event، فإنه يستجيب للحدث بتنفيذ إجراء يؤدي إلى تغيير حالته.

يعتبر مخطط الحالة هو منظور محلي لكائن يفصله عن باقى النظام، أى أنه يدرس سلوك الكائن منعزلاً عما حوله. ولهذا فهو مناسب لدراسة وفهم سلوك كائن معين، ولكنه غير مناسب لدراسة سلوك النظام بأكمله.

الحدث Event،

الحدث هو أى فعل يؤدي إلى تبعات مؤثرة في كائن آخر. يمكن أن يصاحب الحدث مجموعة من المعاملات Parameters التي تحدد شكل التأثير. يتمثل التأثير في شكل حدوث تغيرات على المتأثر. يمكن تصنيف الأحداث إلى أنواع عديدة منها: أحداث النداء Call Events، وأحداث التغيير Change Events، وأحداث الإشارة Signal Events، وأحداث الوقت Time Events.

الحالة State،

يحتوى مخطط الحالة على مجموعة من الحالات لكائن واحد. يمكننا وصف حالة الكائن من خلال ثلاث خصائص هي:

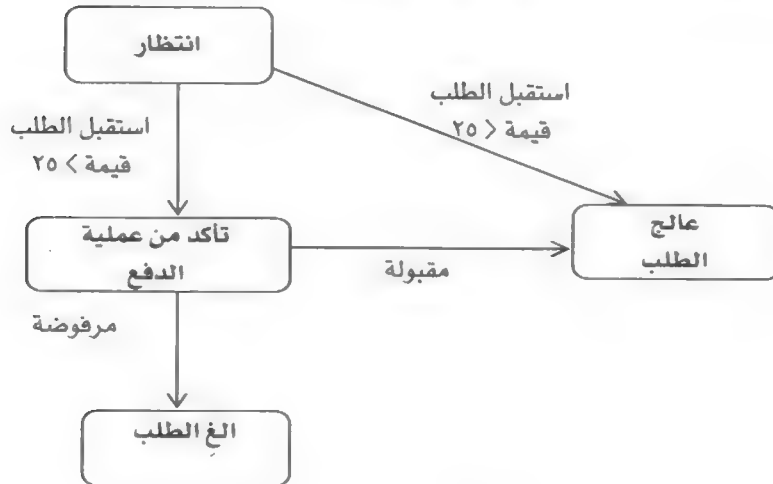
- مجموعة من القيم للكائن.
- فترة زمنية ينتظر خلالها الكائن وقوع حدث معين.
- فترة زمنية يقوم خلالها الكائن بنشاط معين.
- يمكن أن تأخذ الحالة اسماً يصف تأثير الحدث في الكائن. يتم ربط الحالات بعضها ببعض عن طريق خطوط الانتقالات Transitions.

الانتقال Transition:

يحدث الانتقال دائماً من حالة إلى أخرى نتيجة للحدث الذي يؤثر في الكائن ويغير من حالته. يمكننا وصف عملية الانتقال من خلال عدة عوامل، هي:

- زناد الحدث Event Trigger.
- شرط حماية Guard Condition.
- تأثير الحدث Event Effect.
- الحالة الهدف Target State.

يوضح شكل (٢-٢٣) مثلاً لبعض الانتقالات في مخطط الحالة.



شكل (٢-٣٣) الانتقالات في مخطط الحالة

مثال لمخطط الحالة:

يوضح شكل (٢-٢٤) مثلاً لاستخدام مخطط الحالة لوصف الحالات التي يمر بها الكائن «تذكرة Ticket» في نظام حجز وبيع التذاكر للعملاء لحضور عروض ترفيهية. تنحصر الحالات التي يمكن أن يأخذها الكائن فيما يلي:

- متاحة Available.

- محجوزة Locked.

- مبيعة Sold.

يتعرض الكائن «تذكرة Ticket» إلى مجموعة من الأحداث التي تؤدي إلى تغيير حالته مثل:

- حدث وقتي Time Out.

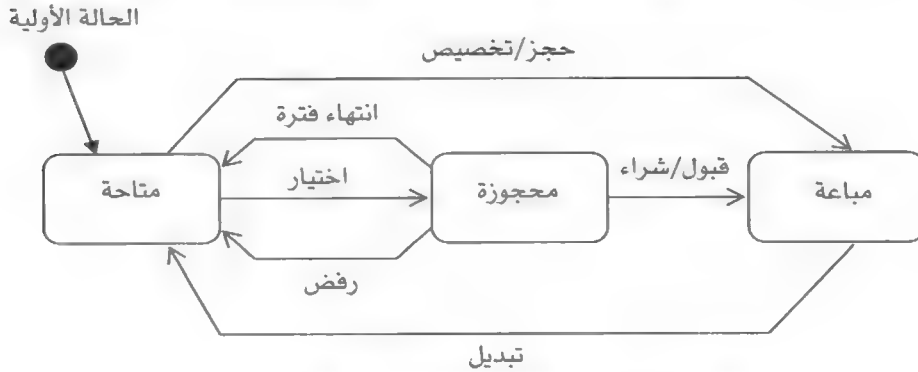
- حدث اختيار Select event.

- حدث رفض Reject event.

- حدث تغيير Change event.

- حدث موافقة Accept event.

فيبدأ الكائن «تذكرة Ticket» دائماً بالحالة «متاحة» ثم ينتقل إلى الحالة «محجوزة» في حالة إتمام عملية الحجز، وينتقل إلى الحالة «مبيعة» عند شراء التذكرة فعلياً. ومن الممكن أن تنتقل حالة الكائن «تذكرة Ticket» من الحالة «مبيعة» إلى الحالة «متاحة» عند استرجاع أو تبديل التذكرة. وينتقل من الحالة «محجوزة» إلى الحالة «متاحة» عند انتهاء فترة الحجز المسموحة بدون شراء التذكرة أو عند رفض الشراء.



شكل (٢-٣٤) مخطط الحالة للكائن Ticket

٨-٢-٢ مخطط النشاط Activity Diagram

يعتبر مخطط النشاط هو أحد أساليب وصف منطق الإجراءات (العمليات) Process Logic أو لوصف تدفق العمل Work Flow. وهو يقوم بدور مشابه لما يقوم به مخطط التدفق Flow Chart مع اختلاف رئيسي بينهما وهو أن مخطط النشاط يسمح بالعمليات أو الوظائف المتوازية. ويتشابه مخطط النشاط مع مخطط التدفق في استخدام نقاط القرار Decision Points والتفرعات Branches في وصف خطوات العمليات. وجدير بالذكر هنا أن الخطوة Step في مخطط التدفق تقابل النشاط Activity في مخطط النشاط. يتم استخدام هذا المخطط بشكل كبير في أثناء عملية التحليل لوصف منطق العمليات أو حالات استخدام النظام.

تستخدم لغة UML مخطط النشاط لوصف خطوات تنفيذ عملية أو سيناريو حالة استخدام داخل النظام في شكل مجموعة من الأنشطة. ويعتبر أسلوب مخطط النشاط بديلاً لأسلوب الوصف النصي لسيناريو محتوى حالة الاستخدام. وهو يتميز بالوضوح وسهولة الفهم خصوصاً في حالات احتواء السيناريو على مجموعة من التفرعات أو التكرارات التي يتم تمثيلها في مخطط النشاط بشكل سهل ومفهوم مقارنة بالشكل النصي. تأخذ المسارات الموجودة بين مجموعة الأنشطة العديد من الأشكال منها ما يلي:

- التوالى Sequence.
- التفرع Branching.
- التوازي Concurrency.

مثال لمخطط النشاط:

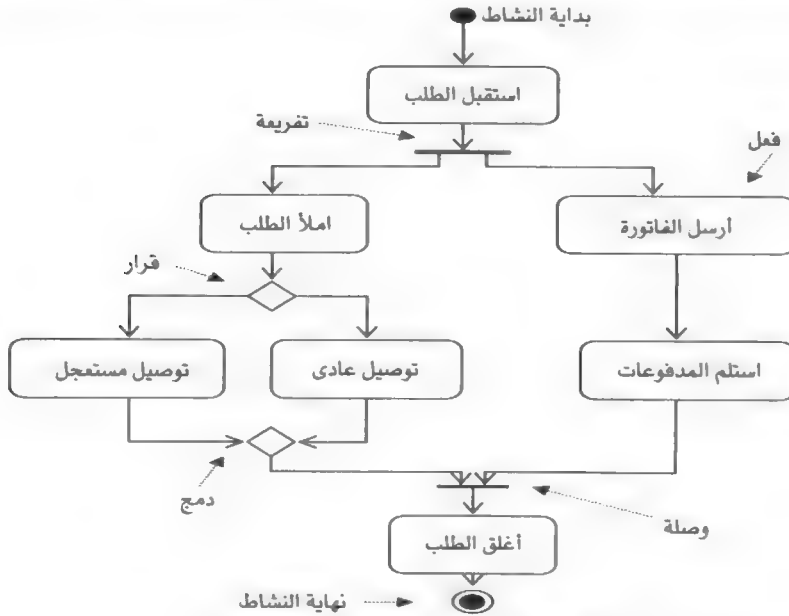
يوضح شكل (٢-٣٥) مثالاً لاستخدام مخطط النشاط في معالجة طلبات العملاء.

نلاحظ في المثال أنه بعد استقبال الطلب يتم اتخاذ مسارين متوازيين هما:

- تعبئة الطلب Fill Order، توصيل الطلب Order Delivery.

- إرسال فاتورة Send Invoice، استلام مدفوعات Receive Payment.

لأن كلا من المسارين لا يعتمد أحدهما على الآخر، فمن الممكن تنفيذهما في وقت واحد. وبعد ذلك يتم التقاء (دمج) المسارين لتكملة الإجراءات بالنشاط Close Order. ففي المسار الأول يتم تحديد نوع الطلب بعد عملية التعبئة، فإذا كان الطلب سريعاً Rush Order، يتم تنفيذ خطوة التسليم السريع، وإذا كان الطلب عادياً، يتم تنفيذ خطوة التسليم العادي. بالطبع يختلف هذا السيناريو من نظام لآخر، ويسمح مخطط النشاط بتمثيل أي أسلوب وكذلك ترتيب النشاطات ومساراتها المختلفة.



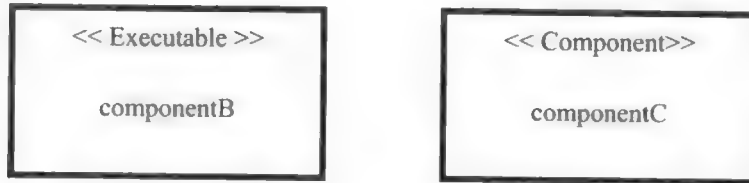
شكل (٢-٣٥) مثال لاستخدام مخطط النشاط.

٩-٢-٢ مخطط المكونات Components Diagram:

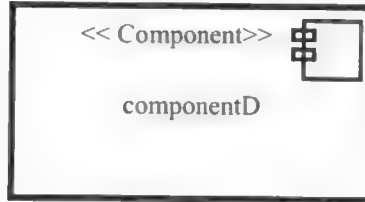
يستخدم مخطط المكونات لوصف مكونات النظام والعلاقات بينها، ولتوضيح ذلك نبدأ بتعريف مكونات النظام.

المكون Component،

المكون هو تمثيل لأحد المنتجات البرمجية للنظام مثل الملفات التنفيذية Executable Files، وواجهات المستخدم User Interface، وملفات التوثيق Documentation Files، وملفات البيانات Data Files وغيرها. يتم تمثيل المكون في لغة UML في شكل مستطيل يحتوى على علامة توضيحية تميزه وهى <<component>> أو علامة تدل على نوع المكون مثل <<Executable>> كما فى شكل (٢-٣٦). ويمكن إضافة رمز المكون المستخدم فى الإصدار الأول للغة النمذجة الموحدة UML1 كما هو مبين فى شكل (٢-٣٧) (رسمى، ٢٠٠٥).



شكل (٢-٣٦) تمثيل المكون فى لغة UML2

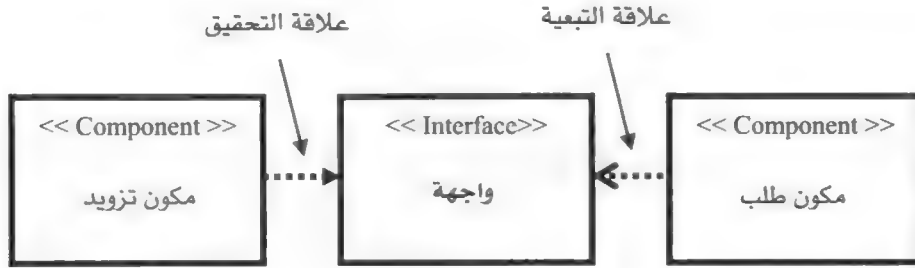


شكل (٢-٣٧) تمثيل آخر للمكون فى لغة UML2

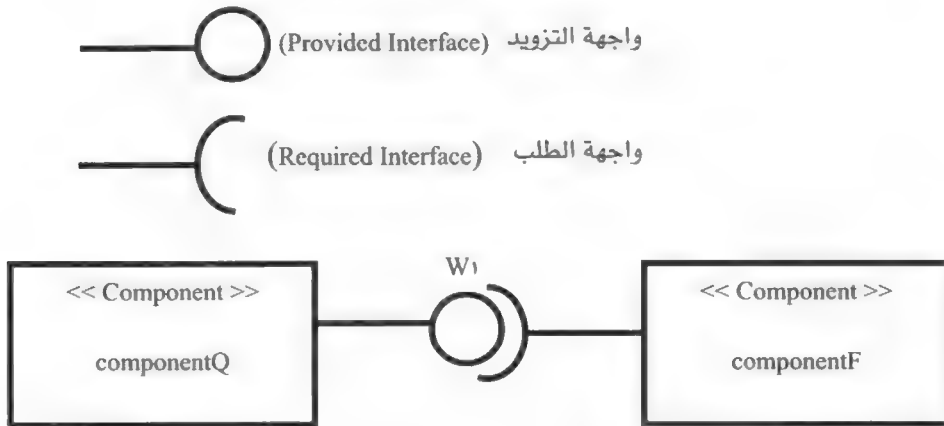
الواجهات والمكونات Components and Interfaces،

الواجهة Interface عبارة عن مجموعة من العمليات التى يمكن من خلالها الوصول أو الاتصال بأطراف أخرى أو مكونات أخرى داخل النظام. يطلق على واجهة المكون الذى يوفر الخدمات واجهة التزويد Provide Interface. وتسمى العلاقة بين المكون

والواجهة «علاقة التحقيق Realization»، أى أن الواجهة هي تحقيق للمكون. ويطلق على واجهة المكون الذى يطلب الخدمات «واجهة الطلب Require Interface»، وتسمى العلاقة بين الواجهة والمكون «علاقة التبعية Dependency». يتم تمثيل هذه العلاقات بطريقتين موضحتين فى شكل (٢-٢٨) وشكل (٢-٢٩).



شكل (٢-٢٨) تمثيل علاقات التحقيق والتبعية باستخدام الأسهم



شكل (٢-٢٩) تمثيل علاقات التحقيق والتبعية باستخدام المكبس والكرة

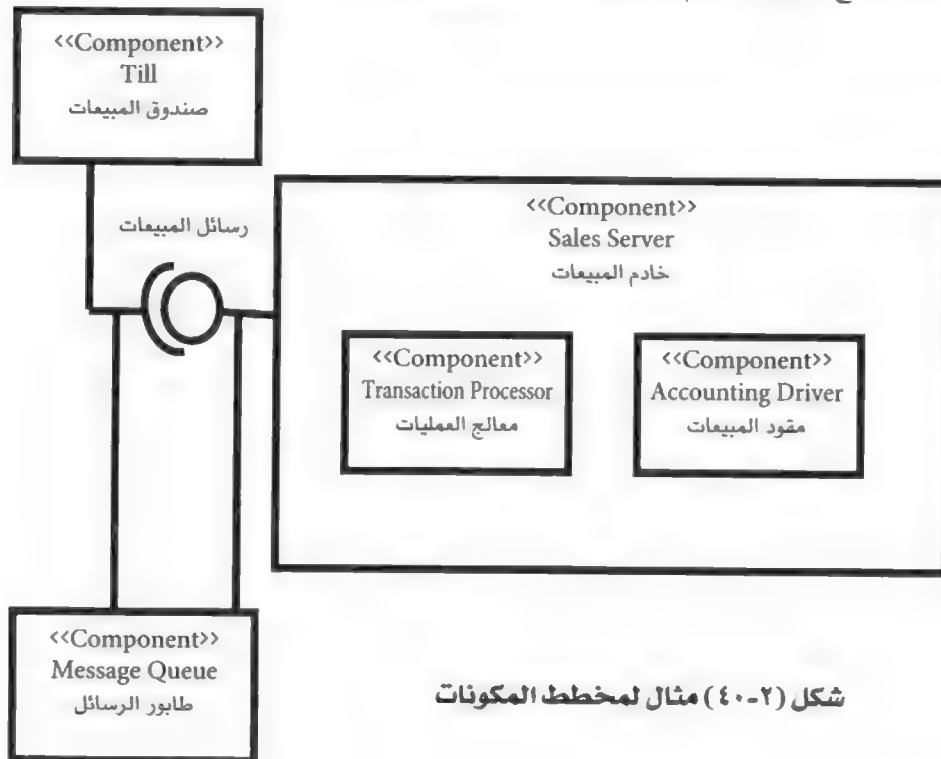
مثال لمخطط المكونات:

تتطلب عملية تصميم مخطط المكونات أن نقوم أولاً بتحديد مكونات النظام، وهى كما ذكرنا سابقاً عبارة عن منتجات برمجية ينتجها أو يستخدمها النظام. بعد ذلك

نقوم بتحديد واجهات المكونات Interfaces، ثم نقوم بوصف العلاقات المختلفة بين المكونات والواجهات (التحقيق والتبعية). يوضح شكل (٢-٤٠) مثالا لمخطط المكونات لأحد أنظمة المبيعات وهو يتكون من المكونات التالية:

- خادم المبيعات Sales server.
- صندوق المبيعات Till.
- طابور الرسائل Message Queue.

نلاحظ في المثال علاقة الربط بين مكون «صندوق المبيعات Till» ومكون «خادم المبيعات Sales Server» من خلال واجهة يطلق عليها اسم «رسائل المبيعات Sales Message». ونلاحظ أن المكون «طابور الرسائل Message Queue» يقوم بعمليتين في المخطط، فهو يقوم بتزويد واجهة «رسائل المبيعات Sales Message» للتعامل مع مكون «صندوق المبيعات Till»، ويقوم أيضا بطلب الخدمة من واجهة «رسائل المبيعات Sales Message» للتعامل مع المكون «خادم المبيعات Sales Server».



شكل (٢-٤٠) مثال لمخطط المكونات

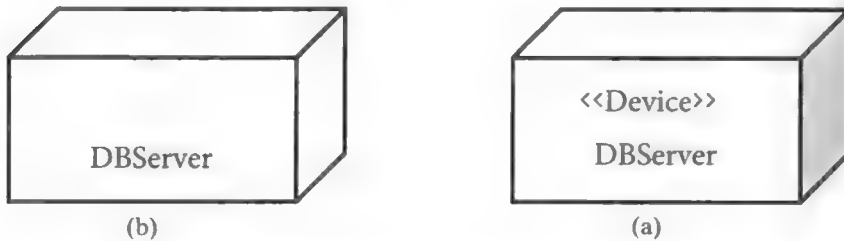
٢-١٠ مخطط النشر:

يقوم مخطط النشر بتمثيل عملية نشر المنتجات البرمجية في أثناء وقت التنفيذ Run-Time من خلال مجموعة من العقد Nodes. تمثل كل عقدة أحد الموارد المادية للعتاد Hardware المستخدم في تنفيذ المنتجات البرمجية للنظام مثل المعالج Processor، والذاكرة Memory، والوحدات الطرفية Peripheral Units. يقوم مخطط النشر أيضاً بتوضيح العلاقات بين المكونات البرمجية Software Components والمكونات المادية Hardware Components ويوضح كذلك كيفية الاتصال بين العقد.

يستخدم مخطط النشر عند الحاجة لمعرفة مكان نشر المكونات البرمجية في النظام. ويستخدم أيضاً في بناء الأنظمة الموزعة Distributed systems؛ لأنه يقوم بتوضيح العلاقات بين المكونات البرمجية الموزعة ومكان نشرها على المكونات المادية، مثل تطبيقات نقاط البيع Point-of-Sales، التي تنفذ من خلال شبكة حاسب موسعة تتضمن مجموعة من العملاء Clients المنتشرين في أماكن متفرقة ومتصلة بخادم مركزي أو أكثر من خادم Servers.

تصميم مخطط النشر:

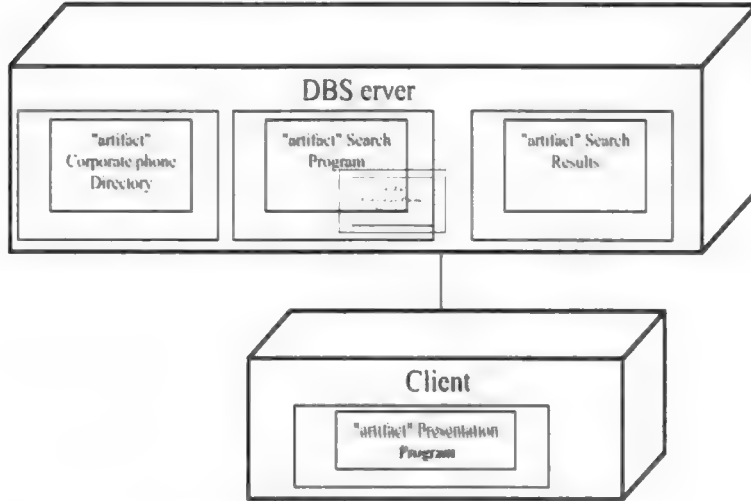
يتكون مخطط النشر من مجموعة من العقد التي تمثل المكونات المادية للنظام. يتم تمثيل العقدة Node في لغة UML باستخدام رمز المكعب وتأخذ اسماً محدداً ويمكن استخدام الكلمة المحجوزة <<Device>>، كما هو موضح بالشكل (٢-٤١) لتمثيل مكون خادم قواعد البيانات DBServer (رسمي، ٢٠٠٥).



شكل (٢-٤١) تمثيل العقدة Node في لغة UML

يتم تمثيل عملية النشر للمنتجات البرمجية من خلال مجموعة من العقد وكيفية الاتصال بينها، كما هو مبين بالشكل (٢-٤٢) الذي يوضح عملية نشر المنتجات البرمجية لنظام «دليل الهاتف Telephone Directory» كما يلي:

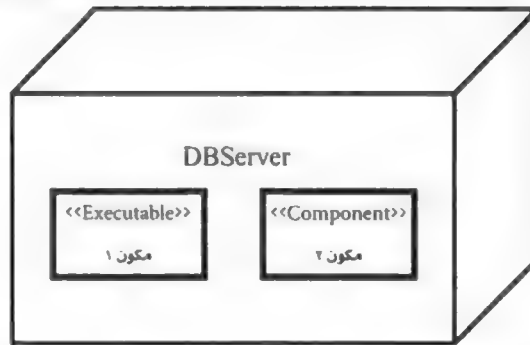
يتم نشر المكونات "Corporate Phone Directory"، "Search Program"، "Search Result" على العقدة، "DBServer" ويتم نشر المكون "Presentation Program" على العقدة "Client".



شكل (٤٢-٢) تمثيل عملية نشر المنتجات البرمجية

دمج مخطط المكونات مع مخطط النشر:

يمكننا دمج مخطط المكونات في مخطط النشر للحصول على مخطط يصف مجموعة المكونات البرمجية التي تقوم بتنفيذها العقد الممثلة للمكونات المادية. وبمعنى آخر فإن العقدة تمثل الموقع الذي يعمل فيه المكون، ولذلك فإنه يتم تثبيت المكون على العقدة، وتسمى تلك العملية «عملية نشر المكونات»، كما هو موضح بشكل (٤٣-٢).



شكل (٤٣-٢) دمج مخطط المكونات في مخطط النشر

مثال لمخطط النشر:

سوف نقوم الآن بتصميم مخطط النشر لنظام إدارة الطلبة Students Administration. ولتمثيل ذلك سوف نبدأ بتحديد العقد المادية للنظام، وكذلك المكونات البرمجية، ثم نقوم بتمثيل العلاقات بينها من خلال مخطط النشر كما يلي:

١- العقد Nodes،

- خادم التطبيقات Applications Server.
- خادم قواعد البيانات DB Server.
- جهاز حاسب رئيسى Main Frame.
- خادم الويب Web Sever.

٢- المكونات البرمجية Components،

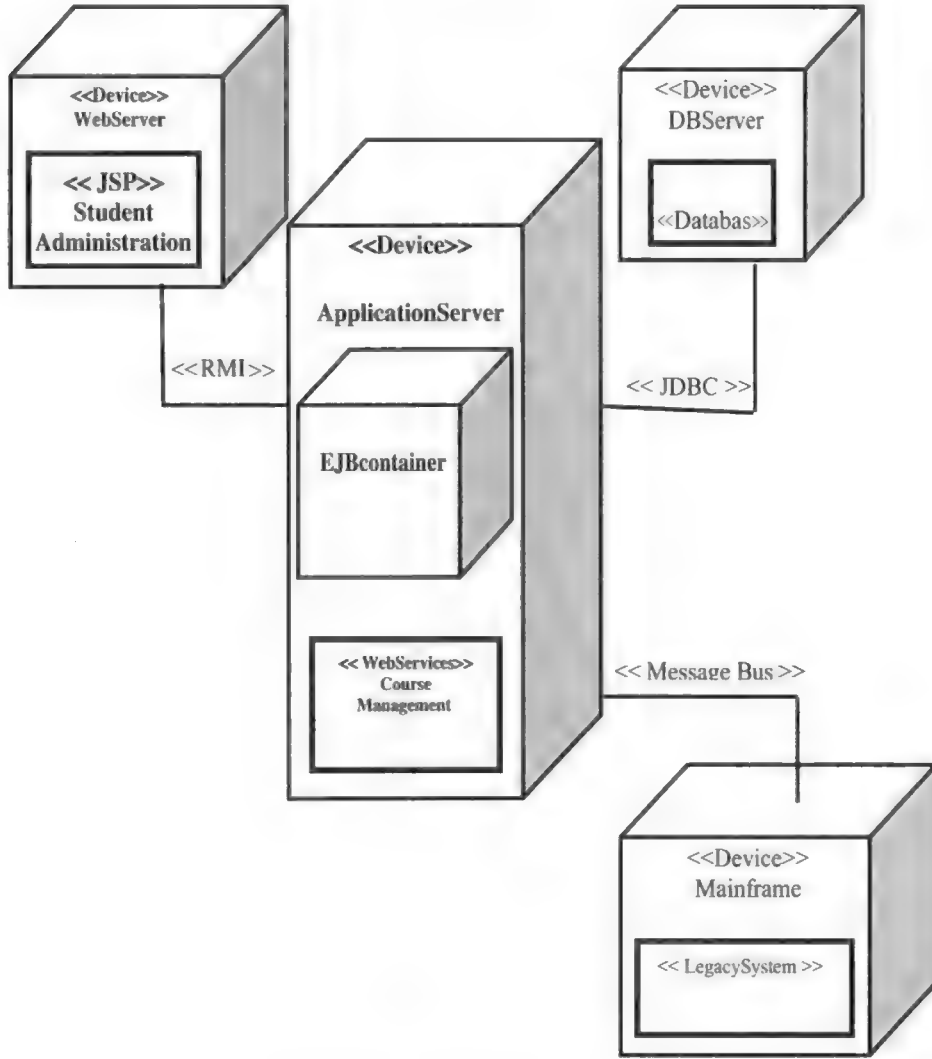
- إدارة الطلاب Students Administration.
- قاعدة البيانات الجامعية University DB.
- إدارة المقررات Courses Management.

٣- الارتباطات بين العقد والمكونات،

- مكون إدارة الطالب مرتبط بعقدة خادم الويب.
- مكون قاعدة البيانات الجامعية مرتبط بعقدة خادم قواعد البيانات.
- مكون إدارة المقررات مرتبط بعقدة الحاسب الرئيسى.

٤- الاتصالات بين العقد،

- تتصل عقدة خادم التطبيقات بعقدة خادم الويب من خلال بروتوكول لغة جافا RMI.
 - تتصل عقدة خادم التطبيقات بعقدة قواعد البيانات من خلال بروتوكول JDBC.
 - تتصل عقدة خادم التطبيقات وعقدة الحاسب الرئيسى من خلال ناقل الرسائل Message Bus.
- يوضح شكل (٤٤-٢) مخطط النشر لنظام إدارة الطلبة.

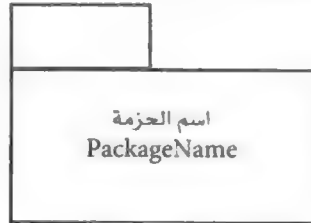


شكل (٢-٤٤) مخطط النشر لنظام إدارة الطلبة

١١-٢-٢ مخطط الحزم Packages Diagram

يعتبر الهدف من تصميم الحزمة Package هو تجميع لبعض مكونات النظام، مثل الأصناف Classes أو حالات الاستخدام Use Cases وهي تكون غالباً مرتبطة وتؤدي

غرضاً محدداً. إذ يتم إحاطة هذه العناصر المجموعة، فى أيقونة مجلد Tabbed Folder، كما هو موضح فى الشكل (٢-٤٥).



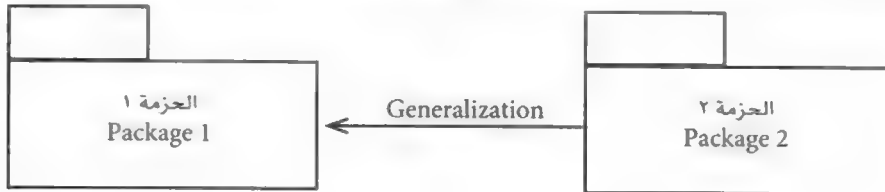
الشكل (٢-٤٥) أيقونة الحزمة (Package Icon)

وعند إعطاء اسم للحزمة Package نكون بذلك قد أعطينا اسماً للمجموعة التى تم تجميعها، وليس شرطاً أن يتم وضع الاسم فى منتصف الأيقونة، يمكن كتابة الاسم فى العلامة المجدولة. ولاستخدام أحد مكونات الحزمة، يجب استخدام الرمز التالى (:)، كما فى الصيغة التالية: PackageName :: PackageElement

وتستخدم الحزمة لتنظيم مكونات أى برنامج تطبيقي (Application Program)، إذ إن الحزمة تمثل مجموعة من الأصناف المترابطة معاً بقوة، ولكنها ضعيفة الارتباط نسبياً بالمجموعات الأخرى. ولا بد من الإشارة هنا إلى أنه يمكن أن ينتمى الصنف الواحد إلى حزمة واحدة فقط، لكن هذا لا يمنع من الظهور فى حزم أخرى أو من الاتصال مع أصناف فى حزم أخرى. يمكن أن ترتبط الحزم معاً من خلال ثلاثة طرق هى:

١- التعميم (Generalizations):

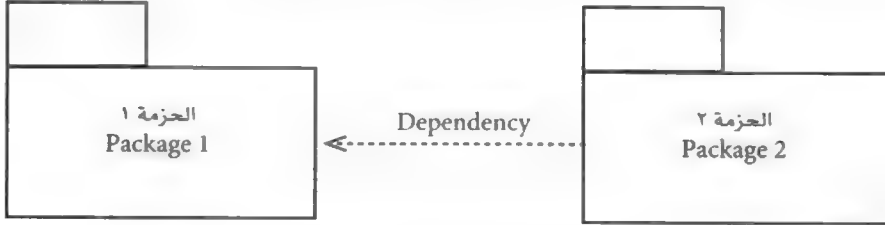
حيث تسمح لحزمة ما باستخدام جميع مكونات حزمة أخرى كما فى الشكل (٢-٤٦).



الشكل (٢-٤٦) علاقة التعميم بين الحزم

٢- التبعية (Dependency):

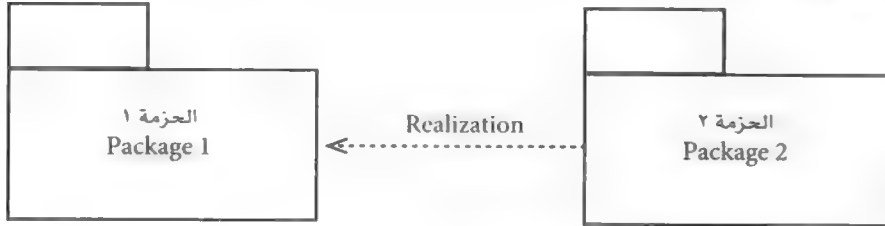
حيث تعتمد مكونات في حزمة ما على مكونات في حزمة أخرى، أي أن أي تغيير يحدث في مكونات الحزمة يؤثر في الحزمة المعتمدة عليها، كما في الشكل (٢-٤٧).



الشكل (٢-٤٧) علاقة التبعية بين الحزم

٣- التحقيق (Realization):

حيث تقوم مكونات حزمة ما بتحقيق أو تطبيق مكونات موجودة بحزمة أخرى، كما في الشكل (٢-٤٨).



الشكل (٢-٤٨) علاقة التحقيق بين الحزم

ما هو مخطط الحزم Packages Diagram؟

إن مفهوم مخطط الحزم غير موجود بذاته في لغة UML. ولكن يتم استخدامه في بناء مخطط الأصناف Classes Diagram أو مخطط حالات الاستخدام Use Cases Diagram للأنظمة الكبيرة الحجم وذلك لغرض تنظيمي فقط نظراً لاحتوائها على عدد كبير من المكونات. إذ يتم إنشاء حزم تحتوي على أصناف أو حالات استخدام، وبعد ذلك تقوم بإنشاء مخطط أصناف تظهر فيه الحزم المخزن فيها الأصناف، ونسمى

المخطط فى هذه الحالة مخطط حزم الأصناف Classes Package Diagram، وذلك لتسهيل عملية فهم مكونات النظام. وكذلك نقوم بإنشاء مخطط حالات الاستخدام، تظهر فيه الحزم المخزن فيها حالات الاستخدام، ويسمى المخطط فى هذه الحالة، مخطط حزم حالات الاستخدام Use Cases Package Diagram.

متى نستخدم مخطط الحزم Packages Diagram؟

تعتبر الحزم من الأدوات الحيوية للمشاريع الكبيرة، لذلك نستخدم الحزم عندما يكون مخطط الأصناف كبيراً جداً، ولا نستطيع وصفه فى مخطط واحد، وعندها نقوم بتجميع مجموعة الأصناف المرتبطة معاً ووضعها فى حزمة واحدة، وتجميع الأصناف التى تشترك فى النوع نفسه من التوريث فى حزمة أخرى، وهكذا. وينصح الكثير من المصممين ومطوري النظم باستخدام لغة UML، أن يتم بناء واستخدام مخطط الحزم فقط عندما نشعر بأنه سيؤدى إلى وصف بسيط ومفهوم ومفيد لبنيان النظام، وأيضاً عندما نريد أن نبقى المكونات أو أجزاء النظام صغيرة قدر الإمكان، للمحافظة على بنيانه وسهولة متابعته (بالمقارنة بمخطط المكونات).

ويظهر ذلك واضحاً، عند استخدام الحزم فى مخطط حالات الاستخدام، إذ تكمن أهميته فى إدارة حجم وتعقيد المكونات الوظيفية للنظام، كما أنه يشكل آلية مهمة وأساسية للتعاون بين مطوري النظام.

مثال لمخطط الحزم Packages Diagram:

سنقوم الآن ببناء مخطط حزم الأصناف لجزء من نظام التسجيل الجامعى.

يتكون هذا المخطط من ثلاث حزم هى:

١- حزمة التقييمات (Marking Package):

تحتوى على جميع الأصناف المرتبطة التى لها علاقة بتسجيل درجات التقييمات للطلاب.

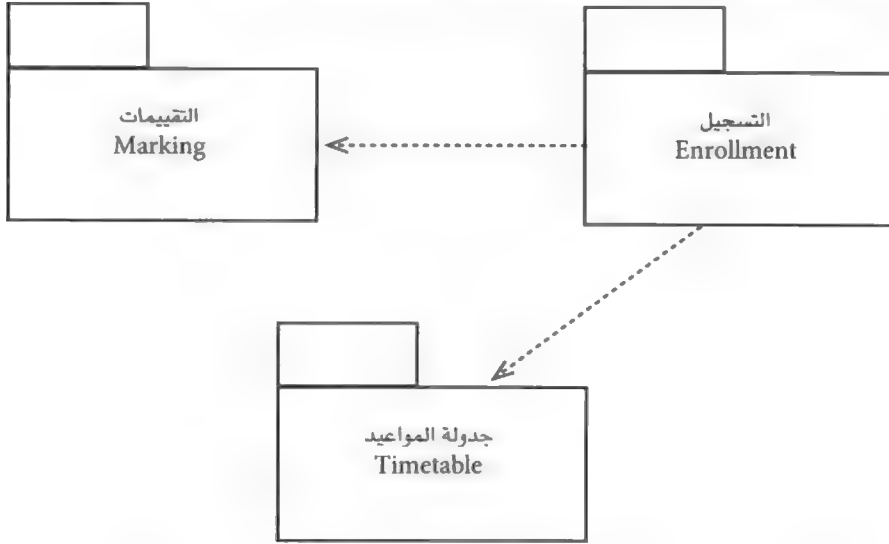
٢- حزمة التسجيل (Enrollment Package):

تحتوى على جميع الأصناف المرتبطة التى لها علاقة بعمليات التسجيل للطلاب.

٢- حزمة جدول المواعيد (Timetable Package):

تحتوى على جميع الأصناف المرتبطة التى لها علاقة بجدولة مواعيد المحاضرات للطلاب.

ويوضح الشكل (٤٩-٢) مخطط حزم الأصناف لهذا النظام.



شكل (٤٩-٢) مخطط حزم الأصناف لجزء من نظام التسجيل الجامعى

نلاحظ من الشكل وجود علاقة التبعية فى المخطط، إذ تعتمد حزمة التسجيل على كل من الحزمتين: حزمة التقييمات وحزمة جدول المواعيد.

الفصل الثالث

أدوات هندسة البرمجيات

Software Engineering Tools

تحتاج عمليات هندسة البرمجيات إلى استخدام العديد من الأدوات في أثناء مراحل التطوير المختلفة وأيضاً في أثناء تنفيذ أنشطة إدارة المشروع. يؤدي الاستخدام الجيد لهذه الأدوات إلى تحسين أداء وإنتاجية الأفراد ومن ثم يعود بالنفع على فريق التطوير نفسه وعلى المشروع كله من خلال رفع كفاءة وجودة المنتجات البرمجية. وعلى هذا فإن استخدام أدوات هندسة البرمجيات يتطلب قدراً معيناً من المعرفة والمهارة. يتمثل الدعم الذي تقدمه هذه الأدوات لعمليات التطوير وإدارة المشروع في استخدامها في أثناء عمليات نمذجة النظام، وبناء وتطوير البرمجيات، ومهام إدارة المشروع. ولذلك فإنه يتم تصنيف أدوات هندسة البرمجيات إلى المجموعات التالية:

- أدوات إدارة المشاريع.
- أدوات نمذجة النظم.
- أدوات برمجة التطبيقات.

ومن المعروف أن هذه الأدوات يتم التحديث فيها بشكل مستمر لتواكب التطور في أساليب ومنهجيات تطوير النظم وكذلك التقنية الحديثة. سوف نستعرض الآن بعض الأدوات المهمة التي تستخدم على نطاق واسع في أثناء تطوير وإدارة مشاريع البرمجيات.

٣-١ أدوات إدارة مشاريع البرمجيات Software Project Management Tools:

تقوم عمليات إدارة مشاريع البرمجيات على استخدام مجموعة من الأدوات والأساليب والطرق المتعلقة بتخطيط ومتابعة إنجاز المهام والعمليات الخاصة بالمشروع. وسوف نتطرق في الفصل القادم لمجموعة من الطرق والأساليب المستخدمة في إدارة المشروع، أما الآن فسوف نركز على عرض لبعض الأدوات المستخدمة في إدارة المشروع. ويجب الإشارة هنا إلى أن تلك الأدوات تستخدم لإدارة أى نوع من أنواع المشاريع سواء

كانت مشاريع برمجيات أو مشاريع أخرى. تركز إدارة المشروع على عمليات تنظيم وإدارة الموارد البشرية والمادية المطلوبة لإنجاز مهام المشروع في فترة محددة ومن خلال ميزانية محددة. وعلى هذا فإن الإدارة الجيدة للمشروع تتطلب التخطيط والمتابعة الجيدة للسيطرة على ثلاثة عناصر أساسية بالمشروع هي:

- الموارد Resources.

- الوقت Time.

- الميزانية Budget.

ويتطلب ذلك توافر مجموعة من الأدوات الفعالة التي تساعد في تنفيذ أنشطة إدارة المشروع المختلفة (مثل جدولة المهام، توزيع الموارد، تقدير التكلفة، مراقبة عمليات التنفيذ، غيرها) بشكل جيد. لقد ظهرت في الآونة الأخيرة بعض الأدوات الحديثة التي تعمل من خلال بيئة متكاملة لإدارة المشروع، حيث تتضمن معظم الأدوات الضرورية الخاصة بتخطيط وتنظيم ومتابعة المشروع. تتسم بعض هذه الأدوات بقدرتها على العمل من خلال شبكة الإنترنت التي تمكن مديري المشروعات الكبيرة من تنفيذ مهامهم بشكل تفاعلي وفي الوقت الفعلي Real Time. فيما يلي عرض لبعض أدوات إدارة المشاريع.

اسم الأداة	عنوان الموقع على الإنترنت
MS Project (Microsoft, 2006)	http://www.microsoft.com/office/project
Manage Pro (ManagePro, 2006)	http://www.managepro.net
eRoom (eRoom, 2006)	http://www.erom.net/eRoomNet
eProject (eProject, 2006)	http://www.eproject.com
Primavera Enterprise (Primavera, 2006)	http://www.primavera.com
Small Worlds (SmallWorlds, 2006)	http://www.thesmallworlds.com
@Risk (Palisade, 2006)	http://www.palisade-europe.com
Risk Radar (ICE, 2006)	http://www.iceincusa.com/products_tools

٣-١-١ جدول المشروع Project Scheduling

هناك العديد من الأدوات التي تستخدم في جدولة أنشطة المشروع ومتابعتها من خلال إعداد مجموعة من المخططات التي تصف الأنشطة ومواعيد بدء كل نشاط والفترة التي يستغرقها النشاط، وكذلك تخصيص الموارد لكل نشاط، وغيرها. يتطلب ذلك تجهيز المعلومات الأساسية التالية:

- تحديد الأنشطة أو المهام Activities or Tasks.

- تحديد الفترة الزمنية لكل مهمة أو نشاط.

- ترتيب تنفيذ المهام.

- تخصيص الموارد لكل نشاط:

بعد ذلك يتم جدولة مهام المشروع باستخدام المعلومات المجهزة في شكل مخطط يدعم عملية متابعة إنجاز المهام خلال مراحل التطوير المختلفة. سوف نستعرض الآن طريقتين من أكثر الطرق استخداماً في أثناء جدولة ومتابعة مشاريع البرمجيات وهما مخطط جانث GANTT chart، ومخطط بيرت PERT chart. تساعد هذه الأدوات مدير المشروع في عمليات ضبط وتعديل جدول المشروع حسب التغيرات التي تطرأ على أنشطة وموارد المشروع في أثناء عمليات التنفيذ.

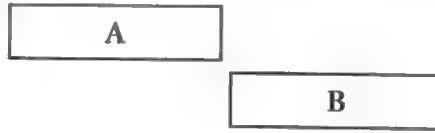
٣-١-١-١ مخطط جانث GANTT Chart

يعتبر مخطط جانث من أكثر الطرق شيوعاً أثناء عمليات جدولة ومتابعة مهام المشروع. يتم فيها تحديد بداية ونهاية كل مهمة وتمثيلها في المخطط حسب ترتيبها الزمني داخل المشروع في شكل مستطيل فارغ. وبعد ذلك يتم استخدام المخطط في أثناء عمليات متابعة تنفيذ مهام المشروع وذلك من خلال تظليل الأجزاء المنفذة فعلياً في كل مهمة ومقارنتها بما هو مخطط مسبقاً في الجدول. ويساعد ذلك مدير المشروع في اتخاذ الأفعال التصحيحية المناسبة لتدارك أي تأخير أو عقبات تحدث لأي مهمة وعدم تراكم تأثيرها في الأنشطة التالية لها.

هناك ثلاثة أشكال لترتيب أنشطة المشروع هي:

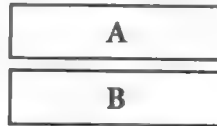
١- توالى Sequence،

حيث يعتمد نشاط معين اعتماداً كلياً على نشاط آخر يكون سابقاً له فى الإنجاز كضرورة لبدء تنفيذه.



٢- توازى Parallel،

فيها يمكن أن يتم تنفيذ نشاطين أو أكثر فى الوقت نفسه، أى أنه لا يوجد اعتماد لأحد الأنشطة على إنجاز نشاط آخر.



٣- تداخل Overlap،

فيها يمكن أن يبدأ نشاط معين بعد بداية نشاط آخر بفترة ولكن لا يبدأ معه فى الوقت نفسه.



أى أنه لا ينتظر لاستكمال النشاط الآخر حتى يبدأ هو فى التنفيذ، ولكن يمكنهما الاشتراك فى فترة زمنية معينة.

تتسم خريطة جانت بالمميزات التالية:

- سهولة التطبيق.
- إظهار الفترة الزمنية لكل نشاط مما يتيح عملية المراقبة أثناء التنفيذ.
- إظهار مدى التداخل الزمنى بين الأنشطة بشكل مرئى وسهل.

مثال لاستخدام مخطط جانت:

سوف نقترح هنا مشروع تطوير نظام معلومات لإحدى المؤسسات التجارية. يقوم المشروع بتطبيق أنشطة دورة حياة تطوير النظم بدءاً من تحديد المتطلبات وانتهاءً بتهيئة النظام. الفترة الزمنية المخصصة للمشروع هي (٢٤) أسبوعاً. والآن سوف نستعرض خطوات جدولة أنشطة المشروع.

١- تحديد أنشطة المشروع:

لا تختلف أنشطة نظم المعلومات كثيراً من مشروع لآخر؛ لأنها تطبق الأنشطة المطلوبة لتحقيق دورة حياة تطوير نظم المعلومات وهي:

- تحديد الاحتياجات Requirement Determination.

- تحليل العمليات Processes Analysis.

- تحليل البيانات Data Analysis.

- تصميم العمليات Processes Design.

- تصميم قواعد البيانات Data base Design.

- تصميم الشاشات Screen design.

- تصميم التقارير Report Design.

- برمجة التطبيقات Applications Programming.

- اختبار البرامج Testing Programs.

- تثبيت النظام System Installation.

٢- تقدير زمن كل نشاط:

هناك ثلاثة تقديرات تستخدم لتحديد الزمن المتوقع لإتمام إنجاز أى مهمة هي:

- تقدير الوقت المتفائل (Optimistic Time (O)، وهو أقل فترة ممكنة لإتمام إنجاز النشاط.

- تقدير الوقت المتشائم (Pessimistic Time (P)، وهو أكبر فترة ممكنة لإتمام إنجاز النشاط.

- تقدير الوقت الواقعي (R) Realistic Time، وهو يمثل أفضل تقدير من وجهة نظر مدير المشروع للفترة الزمنية المعقولة لإتمام إنجاز النشاط.

تستخدم تلك التقديرات لحساب الفترة الزمنية المتوقعة لإتمام إنجاز النشاط كما يلي:

$$ET = (O + 4R + P) / 6$$

نلاحظ من تلك المعادلة أن الوقت الواقعي أخذ وزناً يكافئ أربعة أضعاف وزن كل من الوقت المتفائل والوقت المتشائم نظراً لقربه من الوقت الفعلي للنشاط. يمثل شكل (١-٣) تقديرات الوقت المتوقع لأنشطة المثال الحالي بالأسبوع.

الوقت المتوقع ET	الوقت الواقعي R	الوقت المتشائم P	الوقت المتفائل O	وصف النشاط
٢	٢	٣	١	تحديد الاحتياجات
٣	٣	٥	١	تحليل العمليات
٣	٢	٩	١	تحليل البيانات
٧	٦	١٤	٤	تصميم العمليات
٦	٤	١٨	٢	تصميم قواعد البيانات
١	١	١	١	تصميم الشاشات
٥	٥	٨	٢	تصميم التقارير
٤	٤	٥	٣	برمجة التطبيقات
٨	٧	١٥	٥	اختبار البرامج
٢	٢	٣	١	تنصيب النظام

شكل (١-٣) تقديرات الوقت المتوقع لأنشطة أحد المشاريع

٣- ترتيب أنشطة المشروع:

وهنا يتم تحديد الترتيب الزمني لأنشطة المشروع وعلاقة كل نشاط بالسابق له والتالي له. يوضح شكل (٢-٣) الترتيب الزمني لأنشطة المثال الحالي.

نلاحظ أن الأنشطة رقم (٣)، (٤)، (٥) تعمل بالتوازي، أي أنها تعمل في الوقت نفسه؛ لأن النشاط السابق لها واحد وهو النشاط رقم (٢). ونلاحظ أيضاً أن النشاط رقم (٦) يعتمد على كل من النشاطين (٣)، (٤) أي أنه لا يبدأ إلا بعد الانتهاء من تنفيذ

كلا النشاطين، ولذلك فهو يبدأ فى الخطوة بعد انتهاء النشاط (٤) حيث إنه يحتاج إلى وقت أكبر من النشاط (٣)، وكذلك الحال فى الأنشطة رقم (٧)، (٨)، (١٠).

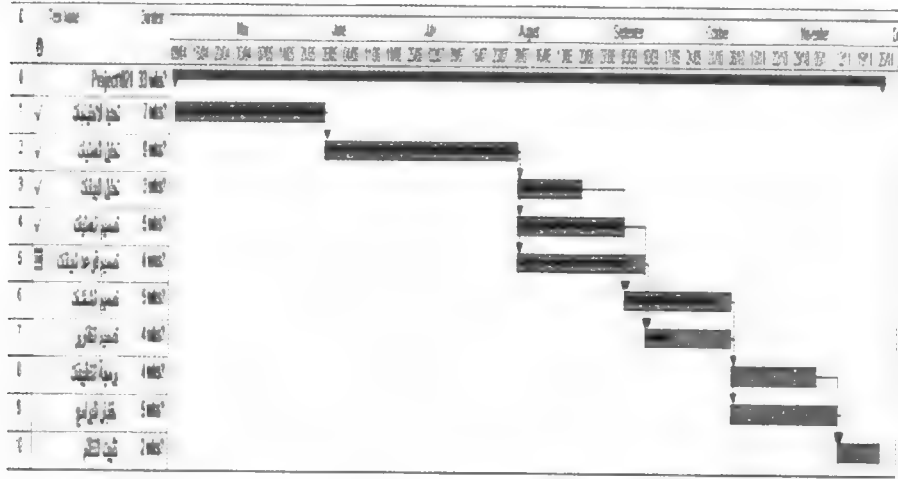
النشاط الحالى	وصف النشاط	النشاط السابق
١	تحديد الاحتياجات	-
٢	تحليل العمليات	١
٣	تحليل البيانات	٢
٤	تصميم العمليات	٢
٥	تصميم قواعد البيانات	٢
٦	تصميم الشاشات	٣ و ٤
٧	تصميم التقارير	٤ و ٥
٨	برمجة التطبيقات	٦ و ٧
٩	اختبار البرامج	٧
١٠	تثبيت النظام	٨ و ٩

شكل (٣-٢) الترتيب الزمنى لأنشطة أحد المشاريع

٤- رسم خريطة جانت للمشروع:

يستخدم كل من التقدير الزمنى لأنشطة المشروع والترتيب الزمنى لها لرسم خريطة جانت لأنشطة المشروع. يوضح شكل (٣-٢) خريطة جانت للمشروع الموضح بالمثل الحالى باستخدام برنامج Microsoft Project (السماك، ٢٠٠٤).

Company	معهد الإدارة العامة
Current Date	٢٠٠٦/١٠/٨
Title	مثال على إدارة مشاريع البرمجيات
Project Start	٢٠٠٦/٠٤/١١ ٠٨:٠٠:٠٠ ص
Project Finish	٢٠٠٦/١١/٢٧ ٠٥:٠٠:٠٠ م
Complete %	%٧٠



شكل (٣-٣) استخدام برنامج Microsoft Project لإعداد مخطط جانت

٣-١-١-٣ مخطط بيرت PERT Chart:

هو أسلوب آخر لتمثيل جدولة نشاطات المشروع بوصف الترتيب المنطقي والزمني لأنشطة المشروع وعلاقة كل نشاط بالنشاطات الأخرى من ناحية اعتماده عليه كلياً أو جزئياً أو عدم الاعتماد عليه إطلاقاً عند بداية تنفيذ النشاط. يتكون مخطط بيرت من مكونين أساسيين هما مجموعة من العقد Nodes تمثل الأنشطة، ومجموعة من الأسهم Arrows تمثل ترتيب الأنشطة. تحتوى كل عقدة فى المخطط على مجموعة من المعلومات عن النشاط الذى تمثله. يوضح شكل (٣-٤) الشكل العام للعقدة فى مخطط بيرت.

يمكننا حساب قيم الأزمنة الأربعة لكل نشاط كما يلى:

- زمن البداية المبكرة للنشاط = زمن البداية المبكرة للنشاط السابق + زمن النشاط السابق.
- زمن النهاية المبكرة للنشاط = زمن البداية المبكرة للنشاط + زمن النشاط.
- زمن البداية المتأخرة للنشاط = زمن البداية المتأخرة للنشاط التالى - زمن النشاط.
- زمن النهاية المتأخرة للنشاط = زمن البداية المتأخرة للنشاط + زمن النشاط.

زمن البداية المبكرة	زمن النشاط	زمن النهاية المبكرة
اسم النشاط		
زمن البداية المتأخرة	الوقت الفائض	زمن النهاية المتأخرة

شكل (٣-٤) الشكل العام للعقدة في مخطط بيرت

ويمكن حساب زمن البداية المبكرة للنشاط (T) Early Start Time بتجميع الوقت المقدر لكل نشاط من بداية الشبكة من اليسار إلى اليمين حتى نصل للنشاط نفسه. وإذا كان هناك نشاط يعتمد على أكثر من نشاط سابق له، فإننا نحسب زمن البداية المبكرة للنشاط الحالي بناءً على الوقت المقدر لكل نشاط من الأنشطة السابقة له، ثم نأخذ القيمة الكبرى لها. ففي المثال الحالي نجد أن النشاط رقم ٦ يعتمد على كل من النشاط رقم ٢ والنشاط رقم ٤، ولحساب زمن البداية المبكرة للنشاط رقم ٦ فإننا نحسب الوقت المتوقع لاستكمال كل نشاط سابق له، وهو يساوي ٨ أسابيع للنشاط رقم ٢، ويساوي (١٢) أسبوعاً للنشاط رقم ٤. ثم نأخذ القيمة الكبرى للوقت المتوقع لاستكمال النشاطين وهو (١٢) أسبوعاً ليمثل زمن البداية المبكرة للنشاط رقم ٦. وبالمثل نجد أن النشاط رقم ٧ يعتمد على كل من النشاطين ٥ و ٤ ويكون زمن البداية المبكرة له تساوي (١٢) أسبوعاً أيضاً. وكذلك الحال للأنشطة رقم ٨ و ١٠.

ويعتبر زمن النهاية المبكرة للنشاط الأخير في المشروع هو تمثيل للفترة الزمنية المطلوبة لاستكمال جميع أنشطة المشروع بدون تأخير وهي تساوي ٢٧ أسبوعاً للمشروع الحالي.

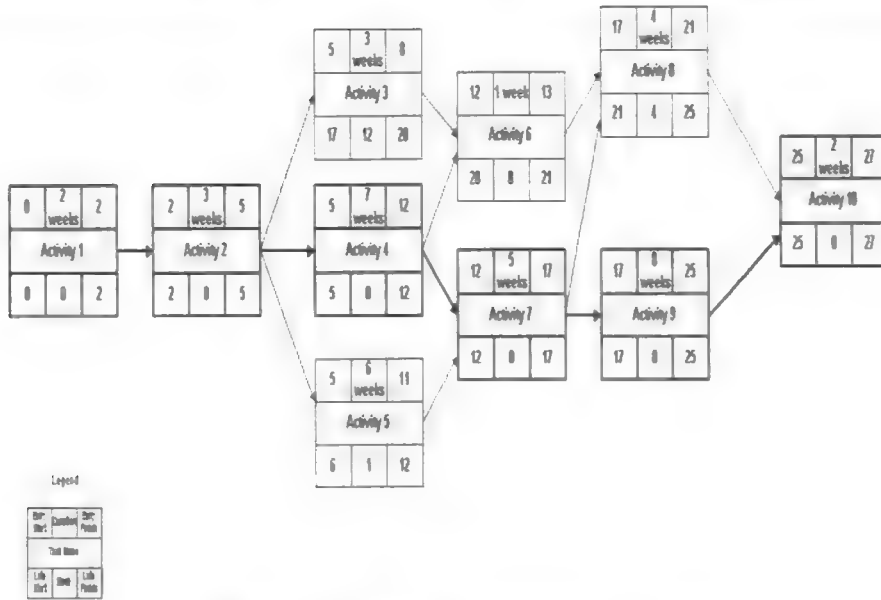
ولحساب زمن البداية المتأخرة لأنشطة المشروع (T) Late Start Time، نبدأ أولاً بالنشاط الأخير في شبكة المشروع ونقوم بإسناد قيمة زمن البداية المتأخرة للنشاط تساوي قيمة زمن البداية المبكرة له (٢٥ أسبوعاً في المشروع الحالي). وبعد ذلك نتحرك من اليمين إلى اليسار ونطرح قيمة زمن النشاط الحالي من قيمة زمن البداية المتأخرة للنشاط التالي له في الترتيب. فمثلاً نجد قيمة زمن البداية المتأخرة للنشاط رقم ٩ $25 - 8 = 17$ أسبوعاً، وقيمة زمن البداية المتأخرة للنشاط رقم ٨ $25 - 4 = 21$ أسبوعاً وهكذا. أما النشاط رقم ٧ فإننا نجد أن كلاً من النشاط

رقم ٨ والنشاط رقم ٩ يتفرعان من النشاط نفسه. ولحساب زمن البداية المتأخرة له نأخذ القيمة الصغرى لزمن البداية المتأخرة لكل من النشاطين ٨ و ٩ (وهو الزمن الخاص بالنشاط رقم ٩ ويساوى ١٧ أسبوعاً). وعلى ذلك يكون زمن البداية المتأخرة للنشاط رقم ٧ $17 = 0 - 12$ أسبوعاً. وهكذا لباقي أنشطة المشروع حتى نصل إلى النشاط الأول بالمشروع. وأخيراً نحسب قيمة الوقت الفائض لكل نشاط وهو يمثل الفرق بين زمن البداية المتأخرة للنشاط وزمن البداية المبكرة له $(Ts = Tl - Te)$.

النشاط الحالى	زمن النشاط	النشاط السابق	زمن البداية المبكرة Te	زمن البداية المتأخرة Tl	الوقت الفائض Ts	المسار الحرج
١	٢	-	٠	٠	٠	✓
٢	٣	١	٢	٢	٠	✓
٣	٣	٢	٥	١٧	١٢	×
٤	٧	٢	٥	٥	٠	✓
٥	٦	٢	٥	٦	١	×
٦	١	٤ و ٣	١٢	٢٠	٨	×
٧	٥	٥ و ٤	١٢	١٢	٠	✓
٨	٤	٧ و ٦	١٧	٢١	٤	×
٩	٨	٧	١٧	١٧	٠	✓
١٠	٢	٨ و ٩	٢٥	٢٥	٠	✓

شكل (٥-٣) حساب قيم الوقت المبكر والمتأخر والفائض لكل نشاط

يوضح شكل (٥-٢) قيم الوقت المبكر Te ، والوقت المتأخر Tl ، والوقت الفائض Ts لكل نشاط للمثال الحالى. يتميز مخطط بيرت عن مخطط جانت بأنه يحتوى على معلومات مفيدة فى أثناء عملية مراقبة تنفيذ المشروع مثل الوقت الفائض Slack Time المتاح عند كل نشاط وهو يساعد مدير المشروع فى تحديد الأنشطة الحرجة التى لا يوجد لديها وقت فائض ومن ثم تحتاج إلى عناية كبيرة فى المتابعة. ويوضح شكل (٦-٢) كيفية تمثيل أنشطة المشروع الحالى باستخدام مخطط بيرت $Pert$.



شكل (٦-٣) مخطط بيرت Pert لأحد المشاريع

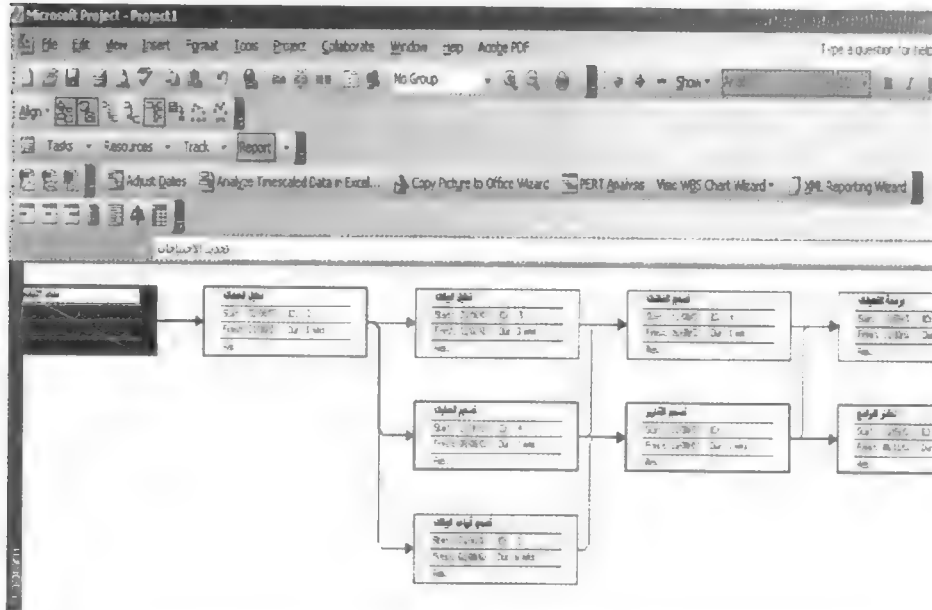
٣-١-١-٣ طريقة المسار الحرج Critical Path Method

يعرف المسار الحرج داخل مخطط بيرت بأنه المسار الذي يمثل أقصر وقت يمكن أن يتم تنفيذ المشروع خلاله. يتم تحديد المسار الحرج من خلال أنشطة المشروع التي لا تحتوي على وقت فائض ($Ts=0$)، وتعرف الأنشطة التي يمر خلالها المسار الحرج بأنها أنشطة حرجة Critical Activities، بمعنى أن أي تأخير يحدث في أي نشاط موجود على المسار الحرج يؤدي بالقطع إلى تأخير المشروع كله. وبالعكس فإن أي نشاط لا يقع على المسار الحرج يحتوي على وقت فائض ($Ts>0$) مما يعني أن التأخير في تنفيذ هذا النشاط لا يؤدي إلى تأخير المشروع كله. ويعرف الوقت الفائض لأي نشاط بأنه الوقت المسموح للنشاط بالتأخير من دون التأثير في وقت اكتمال المشروع.

يمكننا حساب الوقت الفائض لأي نشاط كما يلي:

الوقت الفائض (Ts) = زمن البداية المتأخرة للنشاط (Tl) - زمن البداية المبكرة للنشاط (Te).

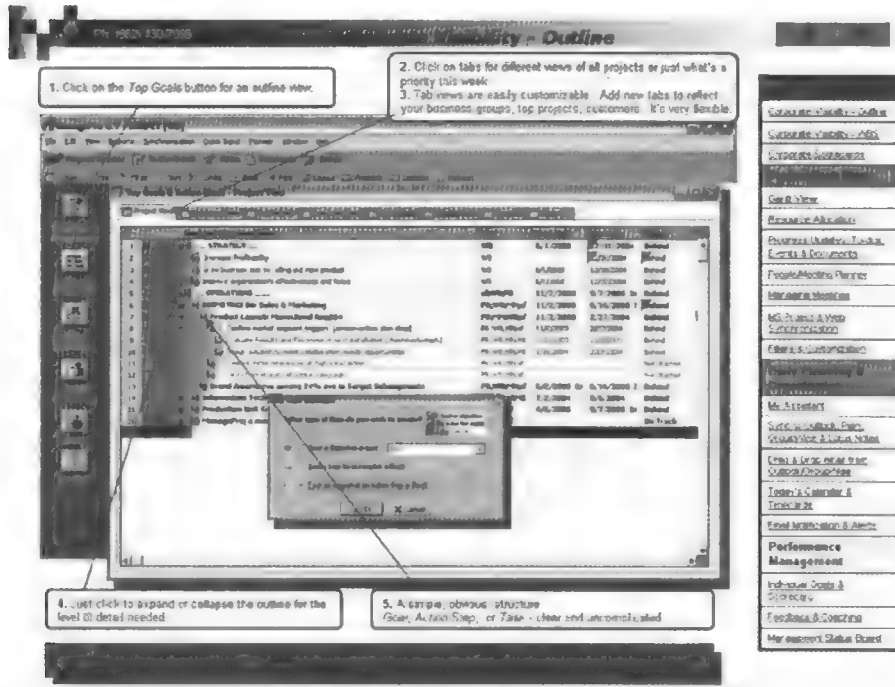
يوضح شكل (٧-٣) مثالاً على استخدام برنامج Microsoft Project لإعداد مخطط بيرت والمسار الحرج لنظام الأعمال الموضح في شكل (٢-٣).



شكل (٧-٣) استخدام برنامج Microsoft Project لإعداد مخطط بيرت

٣-١-٢ إدارة الأداء Performance Management

من الضروري لأي مؤسسة تجارية تريد المنافسة بنجاح في سوق العمل أن تقوم بمراجعة وتعديل أهدافها وخططها تبعاً للتغيرات التي تحدث في متطلبات العميل أو نتيجة استحداث تقنيات جديدة مفيدة لنظام الأعمال أو نتيجة حدوث تغييرات في البيئة الخارجية للمشروع تؤثر فيه. كل ذلك يتطلب من مدير المشروع أن يركز على متطلبات الكفاءة وتحسين الأداء للأفراد القائمين على تطوير المشروع. هناك بعض الأدوات التي تدعم العديد من الخدمات والأنشطة الخاصة بإدارة المشروع ومن ضمنها خدمة إدارة الأداء والتخطيط الإستراتيجي التي تساعد الأفراد وفرق العمل على تخطيط ومتابعة عملية إنجاز الأعمال المكلفين بها. يوضح الشكل (٨-٣) مثالاً لاستخدام أداة ManagePro التي تقدم خدمة إدارة الأداء وخدمات أخرى مثل إدارة الاجتماعات والرسائل الإلكترونية وإنتاج الوثائق، بالإضافة إلى الخدمات الأساسية لإدارة المشروع مثل إدارة الأهداف وتطوير خطة المشروع وجدولة المهام وإدارة الموارد.



شكل (٨-٣) أداة ManagePro لإدارة الأداء

٣-١-٣ إدارة التعاون Collaboration Management

أحياناً يتم توزيع مهام مشروع البرمجيات على فرق عمل تعمل في مواقع مختلفة ومتابعة في الوقت نفسه ويتم الاتصال بين الأفراد المشاركين في عمليات التطوير من خلال شبكة الإنترنت. وهنا يقوم مدير المشروع بإدارة عملية التعاون بين أفراد فرق العمل الموزعة عن بعد باستخدام تقنية شبكة الإنترنت. فيقوم أولاً بتخطيط عملية الاتصال والتعاون بين فرق العمل وتخصيص المهام والمسؤوليات وأيضاً اتخاذ القرارات من خلال الشبكة. ومن ثم يقوم الأفراد المشاركون في فرق العمل باستلام المهام وتسليم منجزاتهم وملاحظاتهم أو آرائهم لمدير المشروع عبر شبكة الإنترنت أيضاً. ويقوم مدير المشروع باستلام جميع المنجزات والمقترحات وتقييمها وإبداء الملاحظات عليها وإجراء التكامل المطلوب بين المنتجات، ثم يقوم بإصدار تقرير بالمنجزات وينشره على صفحات الويب الخاصة بالمشروع ليتم التعرف عليها من

قبل جميع المشاركين والمهتمين بالمشروع. ويقوم مدير المشروع أيضاً بإرسال رسائل إلكترونية Emails لأفراد فرق العمل ليذكّرهم بمواعيد الأنشطة الموزعة وأى معلومات أخرى. وأحياناً يقوم مدير المشروع بعقد اجتماعات عمل Meetings مع فرق العمل من خلال شبكة الإنترنت. هناك العديد من أدوات تطوير صفحات الويب التى تدعم عملية إدارة التعاون مثل برنامج (eRoom (www.eroom.net, 2006.

٣-١-٤ إدارة التكاليف Cost Management

تعتبر عملية إدارة التكاليف من المهام الأساسية لمدير المشروع. فمن المعروف أن ميزانية المشروع تعتمد بشكل أساسى على تكاليف جميع الموارد المستخدمة بالمشروع. ولذلك فإن عملية تحديد تكاليف المشروع بشكل دقيق تساهم فى تقدير ميزانية المشروع بشكل جيد. وبعد ذلك تبدأ عملية إدارة التكاليف لضمان عدم تجاوز الميزانية المقدرة للمشروع. تتركز معظم تكاليف المشروع فى الموارد المطلوبة لإنجاز المهام. تنقسم هذه الموارد إلى نوعين هما:

١- موارد بشرية: تتضمن جميع الأفراد المشاركين فى عمليات تطوير المشروع Stakeholders.

٢- موارد مادية: تتضمن جميع الأجهزة والبرمجيات والتجهيزات ومصاريف الاستهلاك وغيرها.

ويتم تصنيف التكاليف بالمشروع إلى صنفين أساسيين هما:

١- تكاليف استثمارية: هى تكاليف يتم استثمارها قبل بداية المشروع مثل شراء أو تأجير الموقع الخاص بالمشروع وتكاليف تجهيز الموقع بجميع الموارد المطلوبة لبدء المشروع.

٢- تكاليف تشغيلية: هى تكاليف تنشأ مع بدء التشغيل الفعلى للمشروع مثل إيجار أجهزة وأجور الموارد البشرية وتكاليف استهلاك خدمات مثل الكهرباء والاتصالات وغيرها.

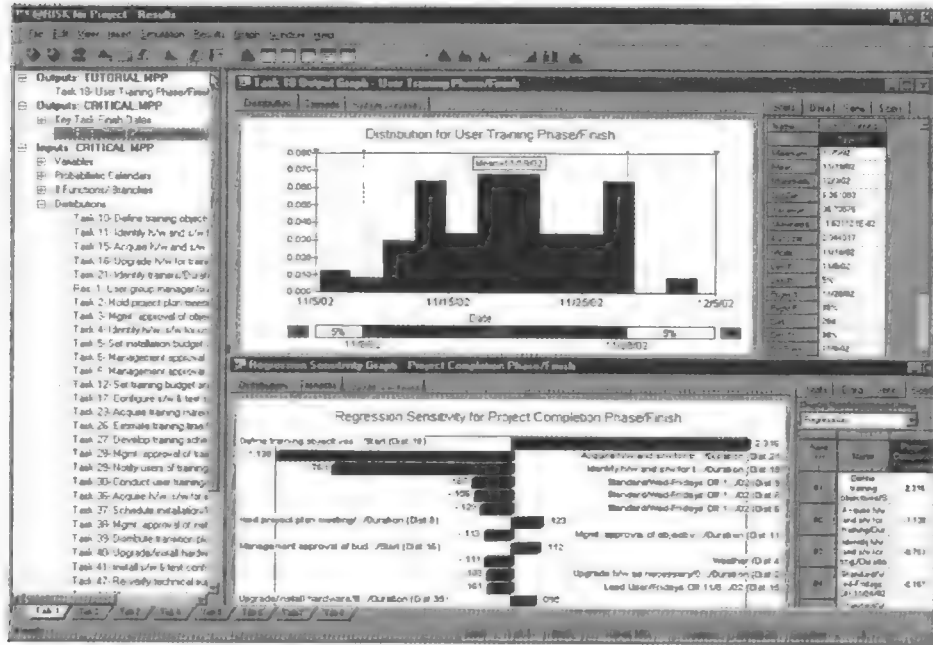
وتتضمن عملية إدارة التكاليف ما يلى:

- تحديد الموارد المطلوبة بدقة.
- تقدير تكلفة الموارد.
- تخصيص الموارد.
- متابعة عملية استخدام الموارد.
- التحكم فى الاستهلاك.
- تقييم الموارد المستخدمة.

هناك عدة طرق تستخدم لتقدير تكاليف المشروع وسوف نستعرضها فى الفصل القادم. وهناك أيضاً العديد من الأدوات التى تستخدم فى عملية إدارة التكاليف (ضمن عمليات إدارة المشروع) مثل برنامجى Primavera Enterprise (Primavera.2006) و eProject (eProject. 2006).

٣-١-٥ إدارة المخاطر Risk Management

تعرف المخاطر بأنها مجموعة الظروف المؤثرة سلبياً فى عمليات تطوير المشروع ومن ثم تؤثر فى جودة المنتج البرمجى. تحتوى جميع أنشطة المشروع على قدر من المخاطر حسب طبيعة النشاط من حيث الوضوح أو الصعوبة والموارد المستخدمة. ولذلك فإن نشاط إدارة المخاطر يعتبر من الأنشطة المهمة فى إدارة المشروع، حيث يقوم مدير المشروع بتحديد الأنشطة التى يمكن أن تحتوى على مخاطرة ويقوم أيضاً بتقييم قيمة المخاطر وإعداد خطة للتغلب عليها عند حدوثها. يستخدم مدير المشروع أساليب الاحتمالات Probability والمحاكاة Simulation لتقدير قيمة المخاطر المتوقعة التى تساعد على اتخاذ القرار الصحيح. هناك بعض الأدوات التى تستخدم بكثرة فى أثناء عملية تقدير المخاطر. فمثلاً يتم استخدام أدوات الجداول الإلكترونية Spreadsheets مثل Microsoft Excel وأدوات قواعد البيانات Databases مثل Microsoft Access. وأدوات المحاكاة Simulation مثل Monte Carlo، بالإضافة إلى أدوات إدارة المشروع فى أثناء عمليات تقدير المخاطر المحتملة بالمشروع. يوضح شكل (٢-٩) مثلاً لاستخدام مخرجات برنامج Monte Carlo من خلال أداة Risk (Palisade, 2003).



شكل (٩-٣) استخدام أداة Risk في تقدير المخاطر

٣-٢ أدوات نمذجة النظام System Modeling Tools

كما ذكرنا سابقاً فإن عملية تطوير مشروع البرمجيات تعتمد على أسلوب النمذجة Modeling في معظم أنشطة التطوير، ويتطلب ذلك توافر مجموعة من الأدوات التي تدعم هذا الأسلوب. هناك مجموعة من الأدوات التي تدعم أسلوب النمذجة المرئية Visual Modeling خلال المراحل المختلفة لعملية تطوير أنظمة البرمجيات. أحياناً يطلق على هذه الأدوات اسم «أدوات هندسة البرمجيات المدعومة بالحاسب CASE» وهي تتميز بدعمها لأسلوب المخزن العام لبيانات النظام أو المستودع Repository الذي يحتوي على جميع المعلومات عن نظام الأعمال، ومن ضمنها النماذج التي تصف النظام من خلال عدة منظورات مختلفة. يساعد هذا المستودع على إمداد جميع أفراد فريق التطوير بالمعلومات المطلوبة. تركز أدوات نمذجة النظم على أنشطة تحليل المتطلبات Requirements Analysis وتصميم النظام System Design. فيما يلي عرض أمثلة لبعض هذه الأدوات:

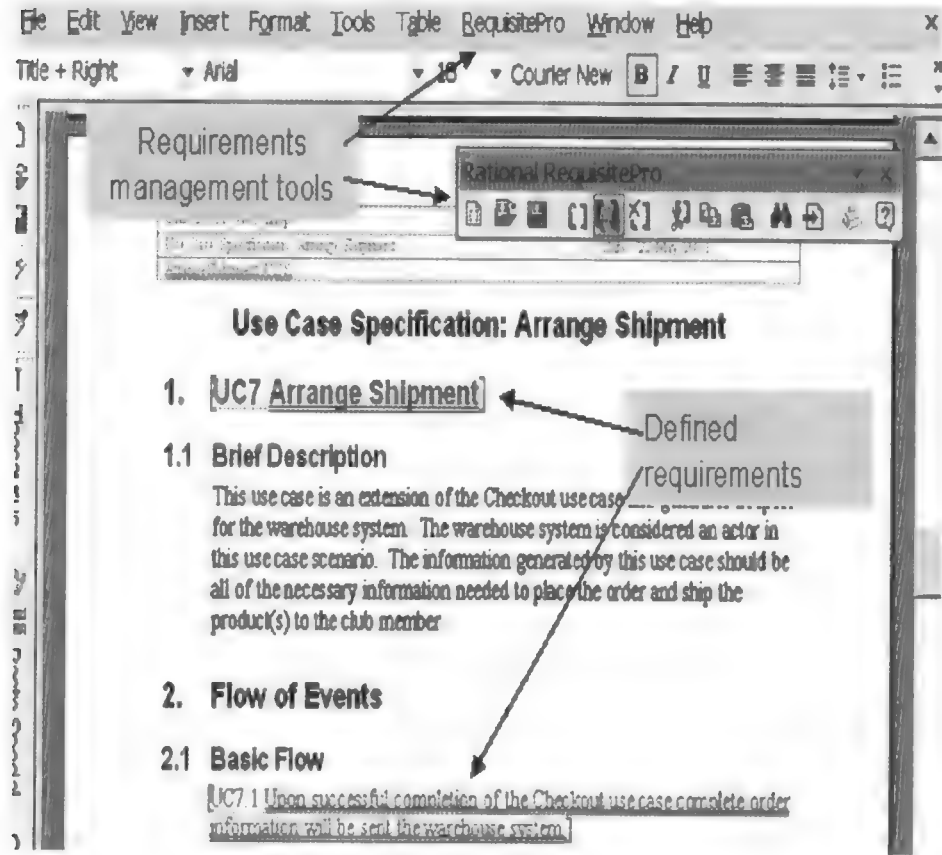
عنوان الموقع على الإنترنت	اسم الأداة
http://www.rational.com	RequisitePro (Rational, 2006)
http://www.telelogic.com	DOORS (Telelogic, 2006)
http://www.sparxsystems.com.au/	Enterprise Architect (Sparx, 2006)
http://www.rational.com	Rational Rose (Rational, 2006)
http://www.gentleware.com	Poseidon (Gentleware, 2006)
http://www.argouml.tigris.org	ArgoUML (ArgoUML, 2006)
http://www.magicdraw.com	MagicDraw (Nomagic, 2006)
http://www.rational.com	Rational SODA (Rational, 2006)
http://www.sybase.com/products/enterprisemodeling	PowerDesigner (Sybase, 2006)

١-٢-٣ إدارة المتطلبات Managing Requirements

المتطلبات هي عبارة عن مجموعة من الجمل النصية التي تصف متطلبات واحتياجات العميل الخاصة بنظام الأعمال الجاري تحليله. تتبع وثائق المتطلبات تنسيقاً وشكلاً معيناً يلائم التمثيل الهرمي Hierarchical Representation. يجب أن تتوافر مجموعة من الخصائص في الأداة التي سوف تستخدم في نمذجة متطلبات النظام مثل:

- تقديم نموذج قياسي يلائم وصف المتطلبات.
- دعم أسلوب التمثيل الهرمي.
- إدارة عملية تغيير المتطلبات.
- إتاحة بيانات المتطلبات لجميع أفراد المشاركين في عمليات التطوير.

يستخدم فريق التحليل هذه المتطلبات لإنتاج نماذج تحليل النظام. ويستخدم فريق التصميم مواصفات المتطلبات لتحديد معمارية النظام. ويستخدم فريق الاختبار هذه المواصفات لإعداد حالات الاختبار للمنتجات البرمجية. ويستخدمها مدير المشروع لإعداد جدول أنشطة المشروع وتحديد الموارد المطلوبة ومن ثم تقدير ميزانية المشروع. يوضح شكل (١٠-٢) جزءاً من وثيقة المتطلبات لإحدى حالات الاستخدام use case التي تم إنشاؤها باستخدام إحدى أدوات إدارة المتطلبات المقدمة من شركة IBM ويطلق عليها اسم RequisitePro (Rational, 2006).



شكل (٣-١٠) وثيقة المتطلبات

يتم تخزين وثيقة المتطلبات في مستودع النظام، وتقوم أداة إدارة المتطلبات بعرض تلك المتطلبات لأفراد فريق التطوير وتسمح لهم أيضاً بالتعديل فيها حسب الحاجة. تدعم الأداة أيضاً عمليات التنقل بين المتطلبات وبين النماذج المرئية المقابلة لها. وتستخدم أدوات النمذجة أسلوب الجداول الإلكترونية لعرض المتطلبات وتداولها.

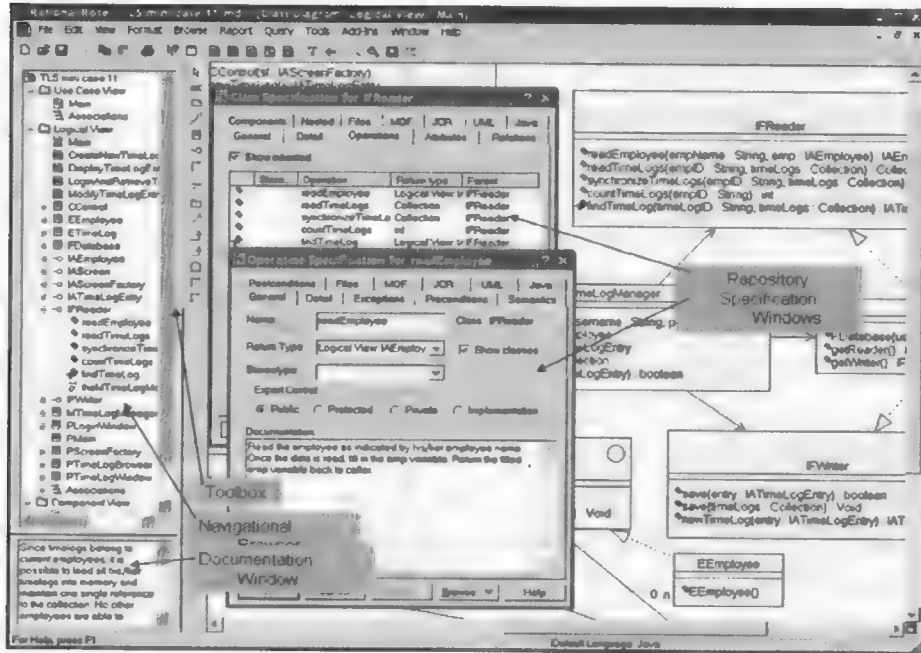
تهتم عملية إدارة المتطلبات بتنظيم عملية ربط المتطلبات بالنموذج الذي يقوم بتمثيلها وبالعناصر المختلفة داخل النموذج مثل الأصناف Classes، والطرق Methods. يؤدي ذلك إلى توافر إمكانية التكامل بين مراحل التطوير المختلفة بدءاً من تحديد المتطلبات مروراً بالتصميم وحتى إنتاج البرامج.

٣-٢-٢ النمذجة المرئية Visual Modeling :

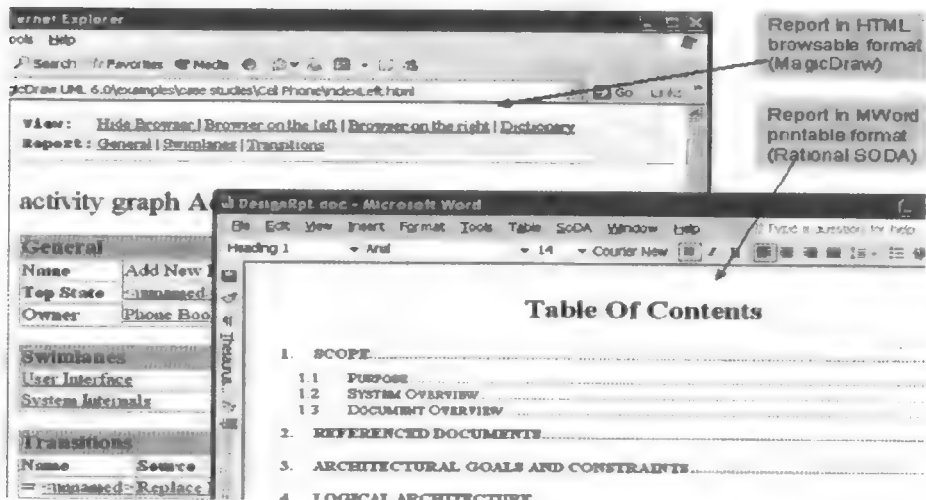
هناك العديد من الأدوات التي تدعم عملية إنتاج النماذج المرئية للنظام. بعض هذه الأدوات يدعم فقط عملية إنتاج المخططات الرسمية للنماذج مثل أدوات الرسم التي تقدمها شركة مايكروسوفت من خلال تطبيقات المكتب MSOffice، والبعض الآخر يدعم عمليات التكامل بين النماذج وإنشاء المستودع Repository وإنتاج قواعد البيانات وتوليد البرامج. يوضح شكل (٣-١١) مثالا لإحدى أشهر الأدوات المستخدمة في نمذجة النظم باستخدام لغة UML المقدمة من شركة IBM وتسمى (Rational, 2006) "Rational Rose". يوضح الشكل مواصفات التصميم لأحد النظم من خلال مخطط الأصناف Class Diagram ومواصفات أحد الأصناف ومواصفات إحدى العمليات له. تمكن الأداة فريق التطوير بالمشروع من إدخال مواصفات المتطلبات الوظيفية لنظام الأعمال من خلال نموذج مخطط حالات الاستخدام وإدخال بيانات محتوى كل حالة استخدام. ومن الممكن أيضاً إنتاج كل من مخطط النشاط ومخطط التتابع لكل حالة استخدام، وبعد ذلك يتم إنتاج مخطط الأصناف بالاستعانة بالمعلومات المقدمة في حالات الاستخدام. تقوم الأداة بعد ذلك باستخدام مخطط الأصناف لإنتاج قواعد بيانات النظام، بالإضافة إلى إنتاج البرامج المبنية على الأصناف بالاستعانة بالمعلومات الموجودة في مخططات التتابع لوصف التفاعلات بين الأصناف. تمكن الأداة أيضاً فريق التطوير من إجراء التعديلات المناسبة على النماذج المخزنة بالمستودع في أى وقت مع إمكانية إعادة إنتاج قواعد البيانات والبرامج المقابلة للتعديلات الجديدة.

٣-٢-٣ توليد التقارير Report Generation :

من الضروري أن يتم إمداد باقي المشاركين في المشروع Stakeholders بمعلومات عن منجزات مراحل التطوير المختلفة. يتم ذلك من خلال مجموعة من التقارير التي يتم إعدادها بشكل يناسب المستخدم للتقرير مثل وثائق نصية أو صفحات ويب أو أشكال توضيحية ورسوم بيانية وغيرها. يوجد العديد من الأدوات التي تدعم عملية توليد التقارير وهي تحتوي على عدد كبير من القوالب Templates يمكن استخدامها في تحديد شكل التقرير. يحدد القالب الشكل العام لهيكل التقرير المطلوب، وبعد ذلك يتم تحديد محتوى التقرير حسب حاجة المستخدم من المعلومات عن النظام. يوضح شكل (٣-١٢) استخدام إحدى أدوات النمذجة وإنتاج التقارير في أشكال مختلفة (Rational, 2006).



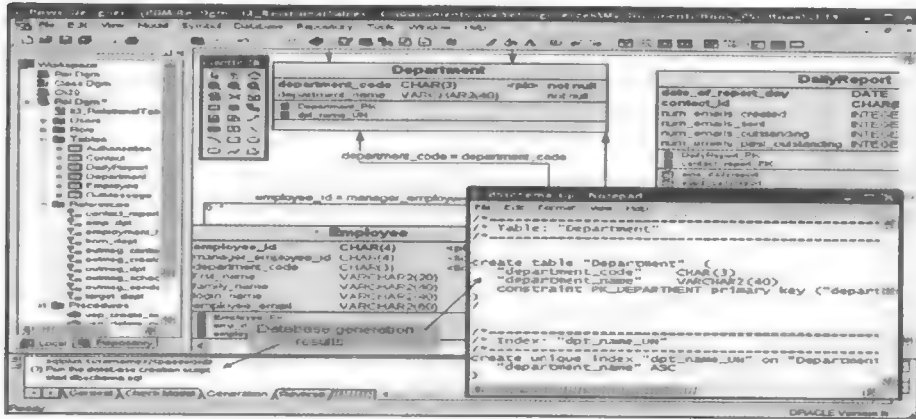
شكل (١١-٣) استخدام أداة النمذجة المرئية Rational Rose



شكل (١٢-٣) استخدام أدوات النمذجة لإنتاج التقارير

٣-٢-٤: نمذجة قواعد البيانات Database Modeling

كما ذكرنا سابقاً فإن جميع بيانات النظام يتم تنظيمها وتخزينها والتعامل معها باستخدام أحد أنظمة إدارة قواعد البيانات. تحتوي قواعد بيانات النظام على مجموعة من الجداول المرتبطة، ويمثل كل جدول كائناً من كائنات النظام أو يمثل علاقة تربط بين بعض الكائنات. يتم تمثيل ذلك من خلال نماذج قواعد البيانات التي تصف مجموعة الكائنات بالنظام والعلاقات بينها. هناك العديد من الأدوات التي تدعم عملية نمذجة قواعد بيانات النظام. تقوم هذه الأدوات بعمليات إنشاء الجداول Creating Tables وتحديد خصائص Attributes كل جدول والمفاتيح الأساسية Primary Keys والخارجية Foreign Keys له، وكذلك مجموعة القيود Constraints والإجراءات المخزنة Stored Procedures بكل جدول، بالإضافة إلى تمثيل العلاقات Relationships بين الجداول. من الممكن أن تستخدم بعض هذه الأدوات نموذج الأصناف Class Diagram بلغة UML لإنتاج نموذج قواعد البيانات، ثم يضاف إليه بعض الخصائص مثل قيود السلامة المرجعية Referential Integrity Constraints. ومن الممكن أيضاً أن يتم استخراج نماذج لغة UML من نموذج قواعد البيانات باستخدام أسلوب الهندسة العكسية Reverse Engineering. يوضح شكل (٣-٢) مثالاً لاستخدام أداة Power Designer المقدمة من شركة Sybase لنمذجة قواعد البيانات لأحد النظم (Sybase, 2006). يوضح الشكل بعض الجداول والعلاقات بينها. والجدير بالذكر أن أداة نمذجة قواعد البيانات تعمل بالتكامل مع أجزاء أخرى داخل نظام إدارة قواعد البيانات Database Management System وهي تنقسم بقدرتها على توليد أكواد هيكل قواعد البيانات schema code والأكواد الخاصة بالإجراءات المخزنة Stored Procedures.



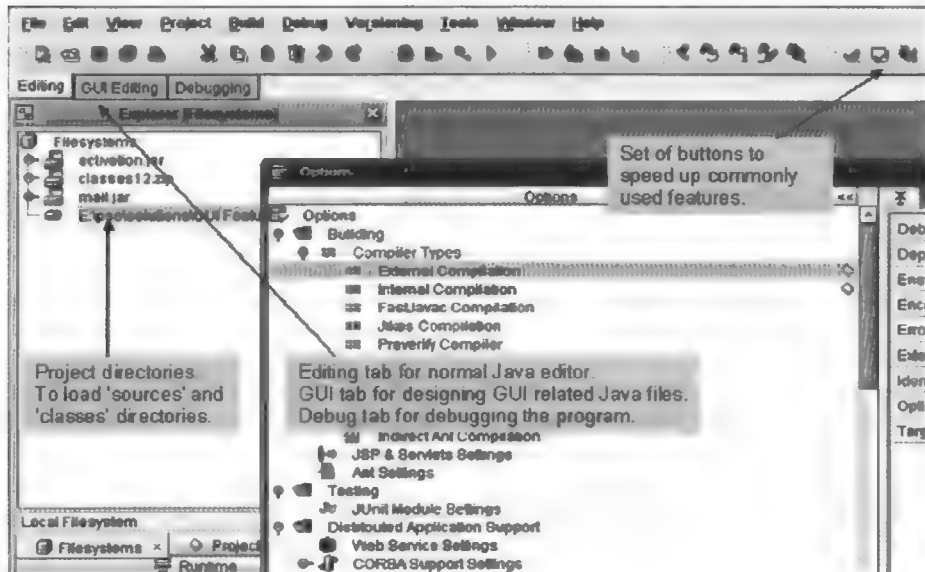
شكل (٣-٢) استخدام أداة Power Designer لنمذجة قواعد البيانات

٣-٣ بيئة التطوير المتكاملة (IDE) Integrated Development Environment:

المقصود ببيئة التطوير المتكاملة هي البيئة التي تساعد المبرمج على إنجاز مهام كتابة البرامج وتدقيقها وتكاملها مع أجزاء برمجية أخرى وإنتاج البرمجيات في الشكل التنفيذي. هناك العديد من الأدوات التي تدعم هذا الأسلوب ولكنها تختلف في بعض الخصائص أو المميزات التي تقدمها للمطورين مثل دعمها لأسلوب نمذجة النظم البرمجية وتكاملها مع أدوات أخرى مثل أدوات إدارة التشكيل Configuration Management Tools لدعم أسلوب فريق العمل أثناء عملية التطوير.

٣-٣-١ كتابة وتنفيذ وتصحيح البرامج Writing, Executing, and Debugging Programs:

الفرض الأساسي من استخدام بيئة التطوير المتكاملة هو مساعدة المبرمجين في كتابة الأكواد بشكل أسرع وتقليل نسبة الخطأ فيها وذلك من خلال توفير بيئة تطوير تدعم أسلوب البرمجة المرئية والبرمجة الموجهة بالكائنات وتكاملها مع كل من نماذج لغة UML ونظم إدارة قواعد البيانات. يوضح شكل (٣-١٤) مثالا لاستخدام بيئة تطوير متكاملة Sun ONE Studio تقدمها شركة Sun Microsystems.



شكل (٣-١٤) بيئة التطوير المتكاملة Sun ONE Studio

تدعم أدوات بيئة التطوير المتكاملة أسلوب البرمجة الموجهة بالكائنات من خلال تقديمها لعدد كبير من الكائنات المعدة سابقاً والجاهزة لإعادة الاستخدام Reusable Objects فى التطبيقات التى يتم تطويرها مع إمكانية ضبطها والتعديل فيها لتلائم طبيعة التطبيق. توفر أيضاً هذه الأدوات مجموعة من المعالجات wizards التى تساعد المبرمج فى إنتاج البرمجيات بشكل أسرع ودقيق. وتتميز بيئة التطوير المتكاملة أيضاً باحتوائها على محرر نصوص حساس للكائنات المستخدمة، يقوم باستعراض خصائصها والطرق التى تقدمها مما يسهل من عملية كتابة الأكواد بشكل صحيح وسريع. وتقوم أيضاً بتقديم المساعدة للمبرمج فى حالة حدوث أخطاء أثناء التعامل مع الكائنات المستخدمة فى التطبيق. وتدعم بيئة التطوير المتكاملة عملية تنفيذ البرامج بعد تدقيقها وترجمتها وربطها بالمكونات المستخدمة من مكتبة الكائنات حتى يتم إصدار الشكل التنفيذى للتطبيق. وتدعم أيضاً عملية تتبع الأخطاء وتصحيحها Debugging من خلال التنفيذ الجزئى للبرنامج حسب رغبة المبرمج (خطوة خطوة أو حتى نقطة معينة أو تنفيذ طريقة معينة Method أو تخطى تنفيذ طريقة معينة فى أثناء التنفيذ).

٣-٣-٢ التكامل مع نمذجة البرمجيات Integration with Software Modeling

تتميز بعض أدوات التطوير الحديثة بتضمينها عمليات نمذجة البرمجيات باستخدام لغة UML فى بيئة تطوير تناسب أسلوب فريق العمل، إذ تمكنهم من التعاون والمشاركة الفعالة فى تطوير النماذج المختلفة للنظام البرمجى واستخراج الأكواد وقواعد البيانات المقابلة لها. ويمكن للمطورين إجراء التعديلات فى النماذج والحصول على الأكواد المقابلة آلياً. ويمكن أيضاً استخدام أسلوب الهندسة العكسية فى استخراج النماذج من الأكواد المكتوبة وقواعد البيانات.

٣-٣-٣ تطوير تطبيقات الأعمال Enterprise Application Development

تستخدم أدوات بيئة التطوير المتكاملة IDE فى تصميم وتطوير ونشر تطبيقات نظم الأعمال الكبيرة والمتوسطة أيضاً. تتطلب عملية التطوير أفراداً ذوى كفاءة وخبرة عالية فى مجالات التصميم والبرمجة وقواعد البيانات، مع الإلمام التام بمفاهيم الكائنات الموجهة واستخدامها فى مراحل التطوير المختلفة. وكذلك التمكن من استخدام أدوات تطوير صفحات الإنترنت، واستخدام تقنية الخادم والعميل Client/Server. تتميز هذه الأدوات أيضاً بإمكانية التكامل مع خادم التطبيقات Application

- Server. تشترك معظم أدوات تطوير تطبيقات نظم الأعمال فى تنفيذ ما يلى:
- تصميم المكونات الأساسية للتطبيق (نماذج Forms أو صفحات ويب Web Pages).
- تطوير محتوى المكونات باستخدام المعالجات المساعدة Wizards.
- إضافة أكواد ضرورية للمكونات المنتجة باستخدام المعالجات.
- اختبار المكونات.
- نشر التطبيق.

هناك العديد من الأدوات التى تستخدم على نطاق واسع فى تطوير تطبيقات نظم الأعمال مثل:

- Oracle JDeveloper, 10g.
- Sun ONE Studio.
- IBM San Francisco.
- Microsoft Visual Studio.

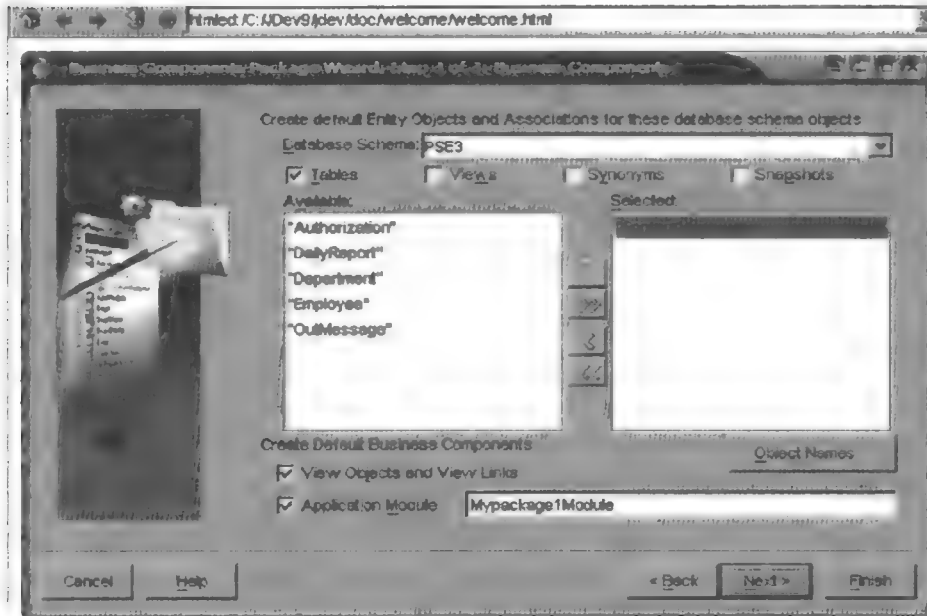
تحتوى معظم هذه الأدوات على معالجات قوية تساعد المطورين على إنجاز جزء كبير من التطبيق بسهولة وسرعة كبيرة. وتقدم أيضاً هذه الأدوات بعض القوالب الجاهزة Templates مثل JSP, EJB, ASP, Web services. وهى تختصر كثيراً من الوقت والجهد المطلوب لتطوير التطبيق. تدعم أيضاً هذه الأدوات الاحتياجات الضرورية لتطبيقات الأعمال مثل:

- استخدام أسلوب النمذجة المرئية Visual Modeling.
- الاتصال بقواعد البيانات Database Connectivity.
- تقديم أنماط مختلفة لواجهات المستخدم User Interface Formats.
- دعم عمليات الهندسة العكسية Reverse Engineering Support.
- التعامل مع خادم التطبيقات Application Server.
- إمكانية العمل فى بيئة الإنترنت Web Based/Enabled Development.

٣-٤ التكامل مع مكونات نظام الأعمال Integration with Business Components

تعتبر مكونات نظام الأعمال Business Components هى الطبقة المتوسطة Middle- Tier داخل نظام البرمجيات المتعدد الطبقات Multi-Tier System. توجد هذه الطبقة بين طبقة واجهة التطبيق Application GUI وطبقة قواعد بيانات التطبيق Application Database. ومن الأفضل أن يتم نشر هذه الطبقة المتوسطة

في خادم التطبيقات Application Server. تعتبر أداة التطوير Oracle JDeveloper من الأدوات التي تدعم أسلوب طبقة مكونات نظام الأعمال الذي يطلق عليه اسم (Business component for Java (BC4J، الذي يساعد على تطوير التطبيقات المبنية على قواعد بيانات. تحتوي طبقة BC4J على كتل بنائية Building Blocks تتكامل مع قواعد البيانات باستخدام أسلوب الاتصال JDBC. وتتضمن طبقة BC4J أيضاً مجموعة ملفات تشكيل مبنية على أسلوب XML تقوم بعملية الربط بقواعد البيانات وإنتاج نوافذ التطبيق Application Windows. تستخدم طبقة BC4J تقنيات JDBC, SQLJ للاتصال بقواعد البيانات واستخراج البيانات المطلوبة لمواجهة التطبيق. يوضح شكل (٣-١٥) إحدى خطوات المعالج الخاص بطبقة BC4J في أثناء الاتصال بقواعد البيانات في أثناء استخدام أداة تطوير التطبيقات المبنية على قواعد البيانات Oracle JDeveloper المقدمة من شركة أوراكل. يتميز أسلوب المعالج بتبسيط عمليات الاتصال بقواعد البيانات واختيار الجداول التي يتم استخدامها في بناء النموذج الخاص بالتعامل مع البيانات، وكذلك تكوين الاستعلامات المناسبة لإدخال واستخراج البيانات من قواعد البيانات، مما يوفر الكثير من الوقت والمجهود لمطوري التطبيقات.



شكل (٣-١٥) استخدام أداة التطوير Oracle JDeveloper

٣-٤ أدوات هندسة البرمجيات المدعمة بالحاسب CASE Tools:

لقد عانت مشاريع البرمجيات في الماضي القريب من اعتماد عملية التطوير على أسلوب ومهارات الأفراد القائمين على تنفيذها وهي تتباين بشكل كبير من شخص لآخر، مما يؤدي إلى صعوبة شديدة في تكامل الأنظمة وصيانتها فيما بعد، نظراً لعدم تقيدها أو اتباعها لمقاييس علمية أو منهجية معينة. وللتغلب على تلك المشكلة؛ تم إنتاج بعض الأدوات التي تستخدم الأساليب القياسية لهندسة البرمجيات وذلك لدعم عملية تطوير النظم. تفضل الآن أغلب المؤسسات استخدام أدوات هندسة البرمجيات المدعمة بالحاسب في تطوير وإدارة أنظمتها لتحقيق العديد من الأغراض مثل:

- تحسين جودة النظام المطور.
- إنجاز النظم في وقت أقل من الطرق التقليدية.
- تسهيل وتحسين عملية الاختبار باستخدام أسلوب التدقيق الآلي للبيانات.
- تحسين عملية التكامل بين أنشطة المشروع.
- تحسين جودة عملية توثيق النظام وضمان اكتمالها.
- تسهيل عمليات الصيانة للنظام.
- المساعدة في اعتماد معايير قياسية لعملية التطوير.
- المساعدة في إدارة المشروع.
- وتتمثل معوقات استخدام أدوات هندسة البرمجيات المدعمة بالحاسب فيما يلي:
- التكلفة العالية لشراء تلك الأدوات.
- التكلفة العالية لتدريب فريق التطوير على استخدامها.
- ضعف درجة الثقة في منتجاتها من قبل بعض المؤسسات.
- عدم اتباع المعايير القياسية للمنهجيات المتبعة داخل المؤسسة.
- ويتمثل تأثير استخدام أدوات هندسة البرمجيات على جميع الأفراد المشاركين في تطوير وإدارة مشاريع البرمجيات فيما يلي:
- تساعد أفراد التحليل والتصميم على إنتاج جزء كبير من المهام الروتينية بشكل آلي بناءً على المواصفات الأساسية التي تم إدخالها، مما يوفر الوقت والمجهود.

- تساعد كثيراً أفراد فريق البرمجة، إذ إنها تقوم بإنتاج الأكواد المقابلة لمواصفات التصميم، ويقتصر دور المبرمجين على تكامل وصيانة المكونات البرمجية المنتجة وإضافة بعض التحسينات الشكلية عليها.
- تساعد مستخدمى النظام ليصبحوا أكثر فاعلية مع مطورى النظام فى أثناء عمليات تطوير البرمجيات المطلوبة ومراجعتها وإبداء الملاحظات عليها.
- تساعد العميل (صاحب المشروع) ليكون أكثر فاعلية مع مدير المشروع فى أثناء عمليات التخطيط الإستراتيجى لمشروع الأعمال.
- تساعد قائد كل فريق من فرق التطوير بالمشروع على القيام بدور أكبر فى قيادة ومتابعة إنتاج كل فرد من أفراد الفريق وكذلك عملية التكامل بين منتجات الفريق.
- تساعد مدير المشروع على إدارة المشروع بشكل أفضل ومتابعة تقدم العمل بالمشروع أولاً بأول والتحكم فى المتغيرات والموارد الخاصة بالمشروع، مما يساهم فى رفع درجة نجاح مشروع الأعمال فى تحقيق أهدافه.

مكونات أدوات هندسة البرمجيات المدعمة بالحاسب؛

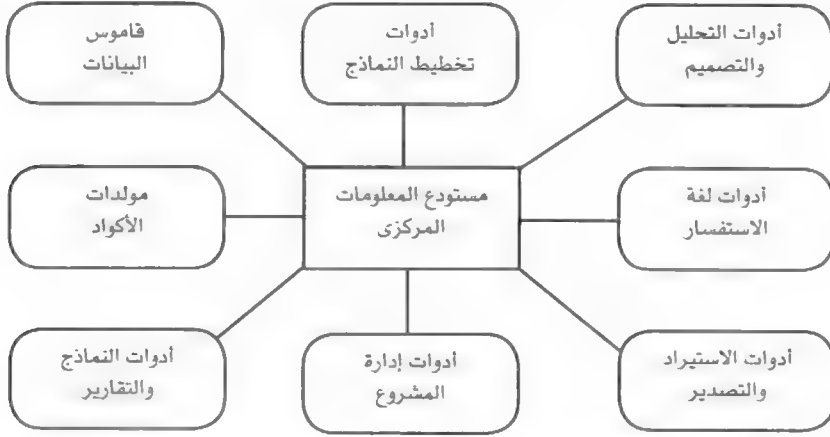
تدعم أدوات هندسة البرمجيات معظم أنشطة تطوير وإدارة مشاريع البرمجيات، فهناك أدوات تستخدم فى مرحلة تعريف المشروع واختياره، وهناك أدوات تستخدم فى أثناء مرحلة التخطيط، وهناك أدوات تستخدم لإنتاج النماذج التخطيطية لكل من عمليات النظام وبياناته، وكذلك عملية توليد البرامج، وإنتاج نماذج واجهات استخدام النظام والتقارير، بالإضافة إلى التوثيق الآلى لجميع مراحل التطوير. يمكننا تنظيم تلك الأدوات فى شكل طبقتين: تسمى الطبقة الأولى «أدوات الطبقة العليا Upper CASE Tools»، وتسمى الطبقة الثانية «أدوات الطبقة السفلى Lower CASE Tools»، كما هو مبين بالشكل (٢-١٦). تحتوى الطبقة العليا على أدوات إدارة المشروع وأدوات التحليل وأدوات التصميم، فى حين تحتوى الطبقة السفلى على أدوات البرمجة والاختبار والتوثيق والتدريب والتثبيت والصيانة (Hoffer et. Al, 2002).

تعتمد أدوات هندسة البرمجيات المدعمة بالحاسب على استخدام تقنية المستودع المركزى Repository لتخزين جميع المعلومات عن كل عملية من عمليات دورة حياة تطوير النظم. ويعتبر هذا المستودع بمنزلة قاعدة بيانات مركزية تحتوى على كل المعلومات الخاصة بمخططات وصف النظام مثل مخططات تدفق البيانات ومخطط العلاقة بين الكائنات ومنطق العمليات وتعريف هياكل البيانات، بالإضافة إلى

خصائص نماذج وتقارير النظام وخصائص واجهات استخدام النظام. ويتم استخدام هذا المستودع في إمداد الأدوات المختلفة بالمعلومات التفصيلية التي تحتاج إليها لإنتاج المخرجات المطلوبة منها بدءاً من خطة المشروع إلى إنتاج الأكواد واختبار وصيانة البرمجيات المنتجة. ويتم أيضاً تسجيل جميع التعديلات التي تحدث في أثناء عمليات التطوير في هذا المستودع. ويوضح شكل (٢-١٧) مكونات معظم أدوات هندسة البرمجيات المدعمة بالحاسب (Sommerville, 2004).



شكل (٢-١٦) طبقات أدوات هندسة البرمجيات المدعمة بالحاسب



شكل (٣-١٧) مكونات أدوات هندسة البرمجيات المدعومة بالحاسب

تحتوى معظم أدوات هندسة البرمجيات على المكونات التالية:

- محرر المخططات Diagram Editor :

يستخدم لإنشاء النماذج التخطيطية مثل نماذج تدفق البيانات ونماذج العلاقة بين الكائنات ونماذج حالات الاستخدام ومخطط الأصناف وغيرها من المخططات، وذلك بشكل قياسي ويتم تخزينها في مستودع النظام Repository.

- أدوات التحليل والتصميم Analysis & Design Tools :

تقوم بعملية التدقيق للتأكد من صحة واكتمال وتناسق مواصفات نماذج التحليل والتصميم للنظام واستخراج الأخطاء لتصحيحها.

- لغة استفسار Query Language :

تسمح للمشاركين في تطوير النظام بالاستفسار والبحث عن أى جزء من مكونات النظام في مستودع النظام.

- قاموس البيانات Data Dictionary :

يحتوى على معلومات تفصيلية عن جميع مكونات النظام وتكون مخزنة في مستودع النظام Repository، ويقوم أيضاً بتوليد توثيق النظام System Documentations.

- أدوات توليد النماذج والتقارير Forms & Reports Generation Tools :
تقوم بتصميم وجهات استخدام النظام User Interface وتصميم التقارير المطلوبة.
- وسائل توريد واستيراد Import/Export Facilities :
تسمح بتبادل المعلومات بين المستودع وأدوات التطوير الأخرى.
- مولدات الأكواد Code Generators :
تقوم بإنتاج أكواد البرامج وقواعد البيانات آلياً وذلك باستخدام مواصفات التصميم المخزنة في مستودع النظام.
- إدارة المشروع Managing the Project :
هي أدوات تقوم بتنفيذ أنشطة إدارة المشروع مثل تحديد الأهداف والمهام وجدولتها وتنظيم الموارد ومتابعة تنفيذ المهام وتعديل الخطة وغيرها.
تعتبر أداة Oracle Designer المقدمة من شركة ORACLE العالمية واحدة من أشهر أدوات هندسة البرمجيات المستخدمة حالياً في تطوير مشاريع البرمجيات.

الفصل الرابع

إدارة مشاريع البرمجيات

Software Project Management

تعتبر إدارة المشروع من الأمور المهمة جداً عند البدء فى تنفيذ مشروع جديد أو تطوير مشروع قائم. تركز عملية إدارة مشاريع البرمجيات على التأكد من الاستخدام الجيد للموارد المخصصة لإنتاج برمجيات تفى بتطلعات المستفيد، وأن يتم تسليم المشروع فى الوقت المحدد وفى حدود الميزانية المتفق عليها. وعلى هذا فيمكننا تعريف إدارة المشروع بأنها عملية مخططة تقوم على تطبيق مجموعة من المعارف Knowledge والمهارات Skills والأدوات Tools والأساليب Techniques على نشاطات المشروع لتحقيق متطلبات وتوقعات المساهمين فى المشروع وذلك فى حدود تقييد زمنى ومالى.

مراحل إدارة مشاريع البرمجيات The Phases of Software Project Management:

تمر عملية إدارة مشروع البرمجيات بأربع مراحل رئيسية هى (شكل (٤-١)):

١- التخطيط Planning.

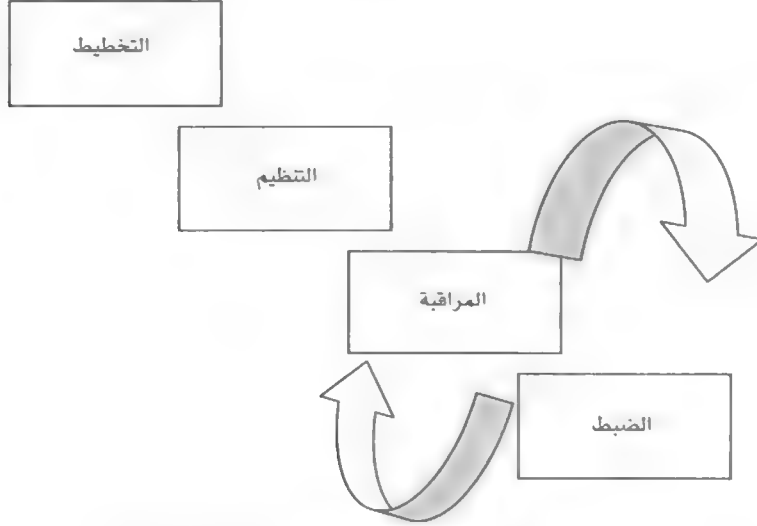
٢- التنظيم Organization.

٣- المراقبة Monitoring.

٤- الضبط Adjusting.

تبدأ دائماً عملية إدارة المشروع بتحديد أهداف مشروع الأعمال ويقوم بها مدير المشروع بالتعاون مع العميل أو صاحب المشروع. وتعتمد هذه العملية على عدة عناصر أساسية هى جودة البرمجيات المطلوبة والفترة الزمنية المتاحة لتنفيذ المشروع والميزانية المقدرة أو المتاحة حالياً لتنفيذه. وبعد ذلك يقوم مدير المشروع بعمليات تخطيط مهام المشروع وتنظيمها ومراقبة تنفيذها والتحكم وضبط عمليات تنفيذ مهام المشروع. يتم ذلك بأسلوب يماثل طريقة الشلال فى عملية تطوير البرمجيات، مع ملاحظة أنه ليس من الضرورى أن تكون جميع أنشطة إدارة المشاريع تتابعية،

ولكن تتداخل أحياناً بعض الأنشطة في مرحلة معينة، أو حتى تتداخل بعض الأنشطة في مراحل مختلفة، مثلما يحدث بين مرحلتى المراقبة والضبط، فنتيجة لعملية المراقبة يمكن أن يقوم مدير المشروع بإجراء تعديل أو ضبط في خطة المشروع الأولية أو في عملية التنظيم أو كليهما (السماك، ٢٠٠٥).



شكل (١-٤) عملية إدارة مشاريع البرمجيات

التخطيط، Planning

تركز مرحلة التخطيط على إعداد خطة المشروع التى تحتوى على مجموعة الأنشطة التالية:

- وصف للمنتج المطلوب (برمجيات) وخصائصه.
- تحديد مهام المشروع.
- جدولة مهام المشروع.
- تحديد الموارد المطلوبة لإنجاز المشروع.
- تقدير ميزانية المشروع.
- تحديد المخاطر التى يمكن أن تواجه المشروع.

التنظيم Organization:

تقوم عملية تنظيم المشروع، بناء على خطة المشروع، بتنظيم الخدمات التي تساهم في دعم عملية تنفيذ خطة المشروع بنجاح، وذلك من خلال تنفيذ مجموعة الأنشطة التالية:

- اكتساب الأفراد ذوي الخبرات المطلوبة للمشروع.
- تحديد المنهجيات والطرق التي سوف تستخدم في تنفيذ عمليات المشروع.
- توفير مجموعة الأدوات التي تدعم تنفيذ عمليات المشروع.
- تحديد المعايير والقياسات التي سوف تستخدم لتقييم مخرجات المشروع.
- ومن المهام الرئيسية لعملية تنظيم المشروع هي عملية التأكد من أن الأفراد القائمين على العمل بالمشروع مؤهلين ومدربين جيداً على استخدام الأدوات المطلوبة، والمنهجيات المختارة، وتطبيق المعايير والقياسات المحددة.

المراقبة Monitoring:

تركز عملية مراقبة مشروع البرمجيات على الأنشطة التالية:

- تجميع البيانات المطلوب قياسها بشكل منتظم.
- تحليل البيانات ومعايرتها.
- تمثيل وعرض البيانات في شكل تقارير دورية.
- إصدار توصيات ومقترحات بناء على البيانات التي تم تحليلها.
- وبذلك فإن عملية المراقبة تهدف أساساً إلى التأكد من مطابقة العمليات الجارية تنفيذها لما هو محدد في خطة المشروع وإمكانية إجراء التعديلات المناسبة في الوقت المناسب.

الضبط Adjusting:

من النادر جداً أن يقوم مدير المشروع بوضع خطة محكمة ويؤسس تنظيمًا لها بدون حدوث أي تعديلات عليها في أثناء مرحلة التنفيذ. هناك تداخل مقصود بين مرحلتى المراقبة والضبط يؤدي إلى إمكانية إجراء التعديلات المناسبة في مرحلة

مبكرة في أثناء تنفيذ المشروع مما يقلل من المخاطر التي تؤثر في نجاح المشروع. أحياناً تكون التعديلات طفيفة ولا تؤثر في سير المشروع حسب الخطة الموضوعة مسبقاً، وأحياناً أخرى تكون التعديلات كبيرة وجوهرية، نتيجة أخطاء جسيمة حدثت في أثناء عملية تحديد المتطلبات أو في أثناء عملية تحديد وتقدير الموارد المطلوبة أو نتيجة حدث طارئ مثل فقد أحد الأعضاء المهرة بفريق العمل، كل ذلك يؤدي إلى إعادة النظر في خطة المشروع وتعديلها بشكل كبير سواء في محتواها أو في الخطة الزمنية أو في الموارد أو جميعها. تحدث دائماً عمليات التعديل والضبط تحت ضغط وظروف غير مريحة، ويتطلب ذلك أن يتحلى مدير المشروع بالثبات والجرأة؛ ليتمكن من اتخاذ القرارات المناسبة لتصحيح الأوضاع في الوقت المناسب بأقل الخسائر مع تجنب التأثير في نجاح المشروع.

٤-١ تطوير خطة المشروع Project Plan Development:

تعتبر عملية تخطيط المشروع بمنزلة الرأس لباقي جسم المشروع. فعملية التخطيط تقوم بتحديد الأفعال (المهام)، وتوقيت تنفيذها، ومن يقوم بتأديتها، ووضع بدائل احتياطية لها، وجدولة المهام للرجوع إليها في أثناء عملية متابعة التنفيذ، واتخاذ أفعال تصحيحية على الخطة لضبط المشروع. ولذلك فإنه يمكننا القول بأن أنشطة تخطيط المشروع تقوم بدور ضابط الإيقاع أو المايسترو لباقي مراحل المشروع. وعلى هذا فإنه إذا لم يتم تخصيص وقت كافٍ لمرحلة التخطيط أو وضع خطة متعجلة وغير دقيقة للمشروع، فسوف يؤدي ذلك إلى فشل المشروع في الغالب. ومن جانب آخر، فإن وضع خطة محكمة ودقيقة للمشروع لا يضمن بالضرورة عدم حدوث مشكلات في أثناء التنفيذ، ولكنها بالقطع تزيد من فرص حل المشكلات والتغلب عليها، مما يقلل أو يحد من احتمالية فشل المشروع.

يقوم مدير المشروع بوضع خطة المشروع بالتعاون مع أغلب المشاركين في المشروع (الممول، العميل، المستخدمين، فريق التطوير،) بقدر الإمكان. وبعد إعداد الخطة، يجب أن يقوم جميع المشاركين في المشروع بمراجعتها والموافقة عليها. ومن الضروري أيضاً أن يكون مدير المشروع على دراية تامة بمبررات الاحتياج للمشروع (مثل تحسين درجة رضا العميل، وتحسين الأداء، وزيادة الإنتاجية، وتقليل التكلفة، ..) حتى يمكنه ترجمة ذلك إلى أهداف واضحة يمكن قياسها وتضمينها في خطة المشروع (Sommerville, 2004).

تحتاج عملية تطوير خطة مشروع البرمجيات إلى إعداد مجموعة من الخطط التفصيلية مثل:

- خطة الجدولة Schedule Plan.
- خطة الميزانية Budget Plan.
- خطة الجودة Quality Plan.
- خطة الاختبار Test Plan.
- خطة تنمية الأفراد People Development Plan.
- خطة إدارة المخاطر Risk Management Plan.
- خطة إدارة التشكيل Configuration Management Plan.

يوضح شكل (٤-٢) مخطط النشاط الذى يمثل أنشطة تطوير خطة المشروع. تبدأ الخطة بنشاط تحديد مجال وأهداف المشروع، وبعد ذلك يقوم مدير المشروع بتجزئة أعمال المشروع، ومن ثم تحديد المهام Tasks والمستلزمات Deliverables والمعالم Milestones للمشروع. وبعد ذلك يقوم مدير المشروع بتحديد الموارد المطلوبة لتنفيذ مهام المشروع التى تتضمن الموارد البشرية والموارد المادية. وبعد ذلك يقوم بتحديد مجموعة المخاطر التى يمكن أن تواجه المشروع فى أثناء التنفيذ، وكذلك تحديد معايير الجودة المطلوبة لمخرجات المشروع. وبناءً على ذلك يقوم مدير المشروع بإعداد الخطط التالية:

- خطة تأسيس وتنمية الموارد.
- خطة الاتصال بين فرق العمل.
- خطة إدارة المخاطر.

وبعد ذلك يقوم بإعداد جدول مهام المشروع وتخصيص الموارد لكل مهمة ويتم استخدام معلومات الجدولة فى تقدير ميزانية المشروع. يقوم مدير المشروع باستخدام جميع المعلومات السابقة فى إعداد خطة المشروع الأساسية Base Line Plan، ومن ثم يقوم بمراقبة التقدم فى تنفيذ مهام المشروع واتخاذ الأفعال التصحيحية المناسبة لضبط عمليات التنفيذ والتحكم فى مخرجات المشروع لتحقيق أهداف خطة المشروع. وفى أثناء عمليات التنفيذ يواجه المشروع غالباً بعض المخاطر والتحديات التى تعوق إنجاز المهام حسب الخطة الموضوعية. وإذا كانت قيود الميزانية والجدول الزمنى للمشروع غير قابلة للتعديل، فيضطر مدير المشروع إلى إجراء تعديلات فى أهداف أو قيود المشروع الأخرى مثل قيود الجودة. أما إذا كانت قيود الميزانية أو

الجدولة غير ثابتة، فإنه يقوم بإعادة جدولة المهام والموارد بالمشروع حسب الاحتياج لتحقيق الأهداف بمواصفات الجودة المطلوبة بالخطوة. سنقوم الآن بعرض موجز لأهم أنشطة تطوير خطة مشروع البرمجيات.

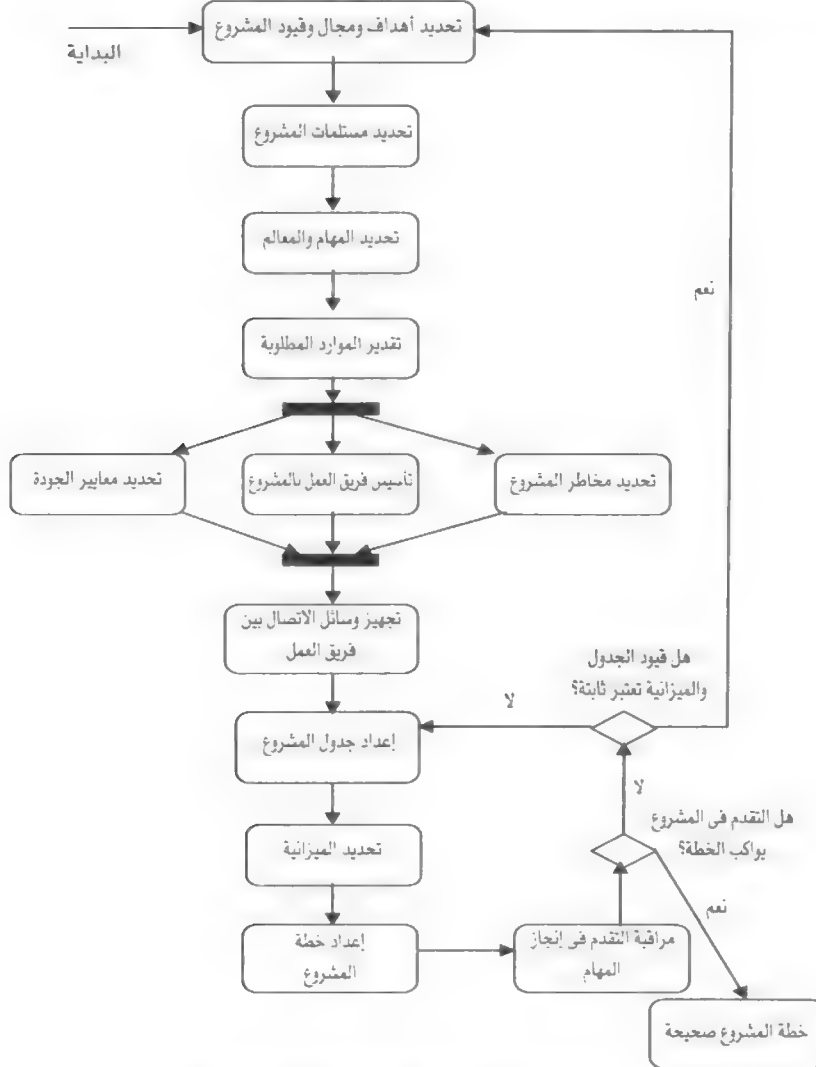
٤-٢ جدول المشروع Project Scheduling

تعتبر عملية جدولة مهام المشروع ومراجعتها من الأنشطة المهمة لمدير المشروع. تتم هذه العملية بعد إتمام عملية تحديد قائمة مهام المشروع. ولأن كل مهمة أو نشاط في المشروع له وقت محدد لبدايته وفترة زمنية محددة لتنفيذه ومن ثم له نهاية محددة، فإن مدير المشروع يقوم بعملية جدولة مهام المشروع بناء على هذه المعلومات بالإضافة إلى مجموعة من القيود التي تنظم عملية الجدولة مثل عملية تخصيص الموارد المطلوبة لكل مهمة. ومن الطبيعي أن تتعرض عملية تنفيذ المهام لبعض المشكلات التي تتطلب تدخل مدير المشروع لحلها وإعادة جدولة المهام حسب الظروف الطارئة حتى لا يؤثر التأخير في إنجاز المهمة في باقى أنشطة المشروع. يقوم مدير المشروع بالاستعانة ببعض الأدوات في أثناء عمليات إدارة المشروع مثل برنامج Microsoft Project الذي يساعده في جدولة المهام وتوزيع الموارد وإعادة الجدولة آلياً حسب المعطيات الجديدة عند حدوث تغييرات. تساعده هذه الأدوات أيضاً في متابعة إنجاز المهام ومراقبة المهام الحرجة بدقة باستخدام مخطط المسار الحرج Critical Path Diagram الذي يحتوي على مجموعة المهام التي يجب تنفيذها في الوقت المحدد لها في الجدول لضمان انتهاء المشروع في الوقت المقدر له في خطة المشروع.

٤-٢-١ المهام والمستلمات Tasks and Deliverables

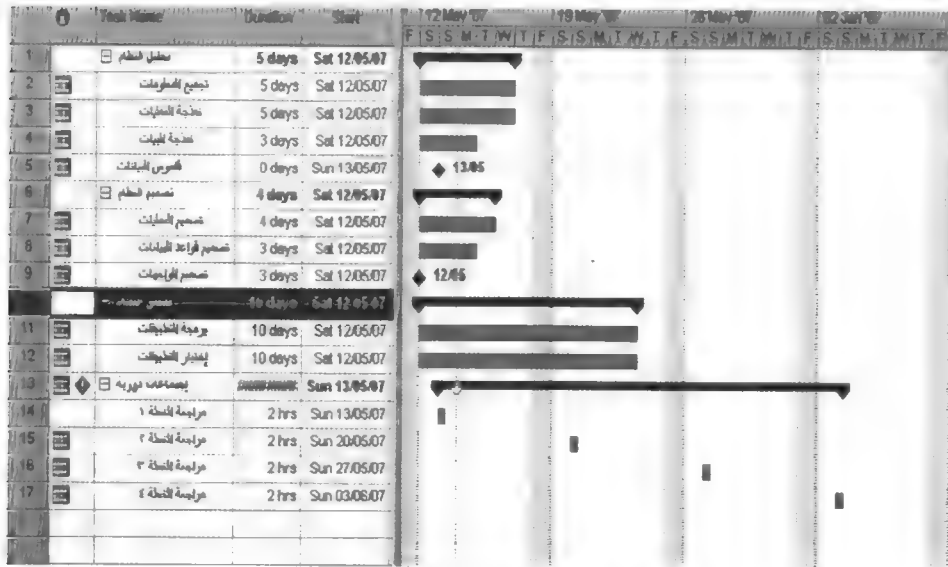
كما ذكرنا سابقاً فإن عملية جدولة المشروع تعتمد على تحديد مهام وأنشطة المشروع. يقوم مدير المشروع - بمساعدة فريق العمل - بتنظيم مهام المشروع من خلال عدة مستويات هرمية تبدأ بالمهام العامة في المستوى الأعلى ثم يتم تفصيلها في المستويات التالية إلى أسفل في شكل مهام فرعية. تعرف هذه العملية بعملية «تجزئة العمل Work Breakdown». يتم تحديد الفترة الزمنية المناسبة لإنجاز كل مهمة وهي تراوح بين عدة ساعات إلى عدة أسابيع حسب حجمها. ومن الأفضل ألا تزيد الفترة المخصصة للمهمة على أسبوعين. وبعد ذلك يبدأ مدير المشروع في إعداد الجدول الزمني للمشروع بدءاً من المهمة الأولى في الترتيب ثم المهام الأخرى التي تليها حتى نصل لنهاية المشروع. وفي الحالات التي تتطلب أن يتم الانتهاء من المشروع في وقت محدد، كأن يكون مرتبطاً بموسم معين يجب أن يكون المشروع

جاهزاً للتشغيل فيه، فإنه يمكن لمدير المشروع أن يبدأ بهذا الوقت ويخصصه لنهاية النشاط الأخير في المشروع ويستمر في جدولة باقى المهام فى الاتجاه العكسى حتى يصل إلى المهمة الأولى فى المشروع، من ثم يتم تحديد الوقت الملائم لبداية المشروع وكذلك حجم الموارد المطلوبة إذا كان الوقت المتاح ضيقاً.



شكل (٢-٤) مخطط أنشطة تخطيط المشروع

يتضمن المشروع بعض المهام التي تؤدي إلى حدث بارز أو تسليم مخرجات جوهرية وهي تعتبر معالم أو علامات فارقة Milestones في المشروع. من الممكن أن تكون هذه المعالم دليلاً على نهاية عملية أو مرحلة أو المشروع نفسه. هذا بالإضافة إلى أن كل مهمة من مهام المشروع يمكن أن تؤدي إلى إنتاج مستلمات Deliverables محددة من مستلمات المشروع. أحياناً يتم تمثيل المعالم والمستلمات في شكل مهام ليس لها فترة زمنية (duration = صفر). يوضح شكل (٤-٣) مثالاً لاستخدام برنامج Microsoft Project لإنشاء قائمة مهام أحد المشاريع. نلاحظ في الشكل أن القائمة تركز على عرض المهام في شكل هرمي وتوضح الفترة الزمنية المخصصة لكل مهمة. وقد تمت تسمية المهام في المستوى الأعلى باسم «Summary Task» ويتم حساب الفترة الزمنية لها آلياً وهي تساوي مجموع الفترات الزمنية للمهام الفرعية لها. ونلاحظ أيضاً وجود مهام روتينية Recurring Tasks تحدث بشكل دوري مثل اجتماع لمراجعة الخطة أسبوعياً لمدة ساعتين.



شكل (٤-٣) قائمة مهام المشروع

٢-٢-٤ جدول المهام Task Scheduling

تعتمد جدولة المشروع على الترتيب الزمني لأنشطة المشروع وعلى الاعتمادية بينها. ولهذا يجب تحديد الاعتمادية بين مهام المشروع قبل عملية الجدولة. غالباً

ما توجد الاعتمادية بين نشاطين أو أكثر نتيجة لطبيعة تنظيم إجراءات العمل أو نتيجة لاشتراك أكثر من مهمة في استخدام الموارد نفسها خصوصاً البشرية. أما المهام التي لا تعتمد على بعضها فيمكن لمدير المشروع أن يجلدولها في الوقت نفسه (أي تعمل على التوازي) أو حسب ظروف الموارد المطلوبة. تقدم أداة Microsoft Project أربعة أنواع من الاعتمادية بين المهام هي:

١- نهاية - إلى - بداية Finish-to-Start:

لا يمكن أن تبدأ المهمة التابعة successor إلا بعد انتهاء المهمة السابقة لها predecessor.

٢- بداية - إلى - بداية Start-to-Start:

لا يمكن أن تبدأ المهمة التابعة إلا بعد بداية المهمة السابقة لها.

٣- نهاية - إلى - نهاية Finish-to-Finish:

لا يمكن أن تنتهي المهمة التابعة إلا بعد انتهاء المهمة السابقة لها.

٤- بداية - إلى - نهاية Start-to-Finish:

لا يمكن أن تنتهي المهمة التابعة إلا بعد بداية المهمة السابقة لها.

وبعد تحديد تلك التبعية بين المهام، فإن أي تعديل يحدث في بداية أي مهمة أو في الفترة الزمنية لها سوف يؤثر آلياً في جدولة المهام التالية لها.

هناك نوعان آخران للتبعية بين المهام هما:

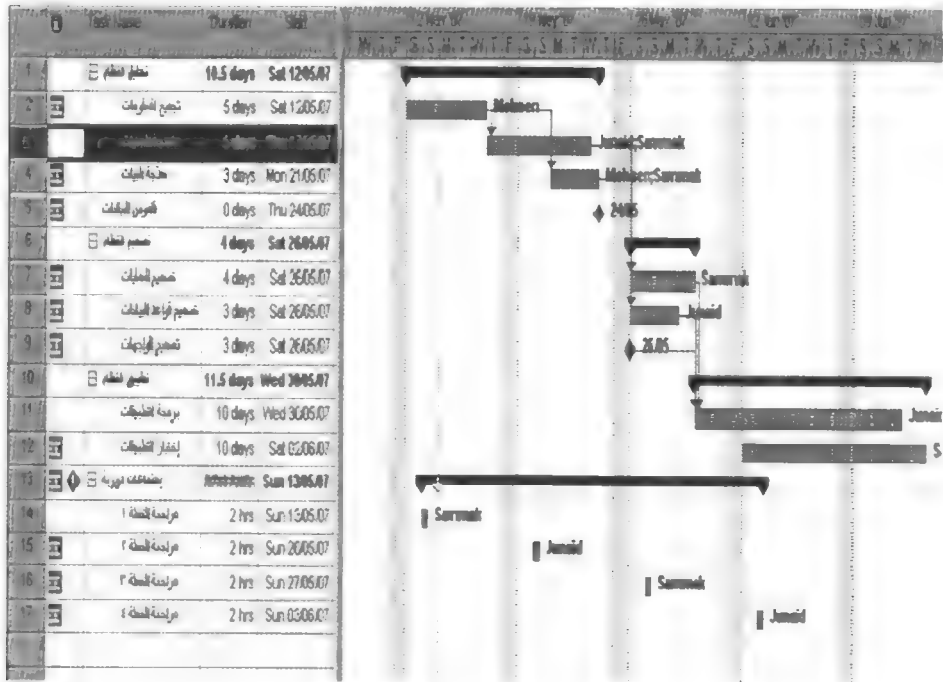
- تبعية التأخير Delay Dependencies:

تحدد فترة زمنية محددة بعد نهاية المهمة السابقة حتى تبدأ المهمة التالية لها.

- تبعية متداخلة Overlap Dependencies:

يحدث فيها تداخل بين المهام، إذ يمكن أن تبدأ مهمة قبل نهاية المهمة السابقة لها بفترة معينة.

الآن وبعد تحديد قائمة المهام وتحديد التبعية بينها، يتم إنشاء جدول مهام المشروع باستخدام أسلوب خريطة جانت GANTT Chart. يوضح شكل (٤-٤) مثالا لاستخدام أداة Microsoft Project في إنشاء الجدول الزمني لمهام أحد مشاريع البرمجيات.



شكل (٤-٤) جدولة مهام أحد المشاريع باستخدام خريطة جانت

٤-٢-٣ تقويمات الموارد Resources Calendars

يتم تصنيف الموارد المطلوبة لإنجاز كل مهمة في المشروع إلى نوعين:

- موارد بشرية:

تصف مجموعة الأفراد القائمين على إنجاز المهمة.

- موارد مادية:

تصف مجموعة الأجهزة والبرمجيات والمواد المطلوب استخدامها لإنجاز المهمة.

يمكن أن تحتاج المهمة إلى أكثر من مورد سواء من كلا الصنفين أو أحدهما. وقبل عملية تخصيص الموارد على المهام، يجب تحديد الموارد المطلوبة ونوعها وكميتها والوقت المطلوب توفيرها فيه وفترة استخدامها. وبعد انتهاء عملية تخصيص الموارد

على المهام، يقوم مدير المشروع بمتابعة وتقييم الموارد المستخدمة. فيقوم بتقييم حجم الإنجاز للموارد البشرية وكذلك معدل استخدام الموارد المادية. وبناءً على ذلك يمكن لمدير المشروع أن يعيد تخصيص الموارد على المهام حسب معدل الإنجاز أو الاستخدام، فيمكنه الاستفادة من الموارد القليلة الاستخدام في إحدى المهام بإعادة تخصيصها لمهام أخرى تحتاج إليها. وكذلك يقوم مدير المشروع بتقييم تكلفة الموارد المستخدمة. تدعم الكثير من أدوات إدارة المشروع إمكانية التعديل في معلومات الموارد المستخدمة في المشروع من خلال استخدام تقويم يطلق عليه «تقويم الموارد Resource Calendar» الذي يمكن مدير المشروع من إدخال أوقات عمل أو استخدام الموارد وكذلك مواعيد التوقف عن العمل كما هو موضح بشكل (٤-٥).

Resource Name	Email Address	Windows User Account	Group	Standard Rate	Overtime Rate
1 Mohsen	mzenaty@ipa.edu.sa			100000	150000
2 Junaid	junaidd@ipa.edu.sa			100000	150000
3 Sammak	sammaka@ipa.edu.sa			100000	150000
4 Khalid	hamedk@ipa.edu.sa			100000	150000

Change Working Time

For: Mohsen based on Standard calendar

Set working time for selected date(s)

Legend:

- ☐ Working
- ☐ Nonworking
- ☐ Edited working hours

On this calendar:

- ☒ Edits to a day of the week
- ☐ Edits to an individual day

Select Date(s):

May 2007

M	T	W	Th	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Set selected date(s) to:

- ☒ Use default
- ☐ Nonworking time
- ☐ Nondefault working time

From: 08:00 To: 12:00

01:00 05:00

Help New Delete OK Cancel

شكل (٤-٥) تقويم الموارد

٤-٢-٤ جدولة المجهود Effort Scheduling

تعتمد جدولة المشروع على تخصيص الموارد على مهام المشروع هي تعتمد بدورها على فترة استخدام الموارد. فمثلاً إذا كان أحد الأفراد يعمل لفترة نصف

الوقت وبدون قيود فى تقويم الموارد، فإن هذا الفرد يكون متاحاً لفترة (٤) ساعات فى اليوم. وإذا كان حجم العمل المكلف به الفرد يزيد على هذه المدة فيعتبر الفرد مكلفاً بعمل إضافي، مما يؤدي إلى تقليل فترة إنجاز المهمة ومن ثم يؤثر إيجابياً فى جدول المشروع. وعلى هذا فإن إجمالى المجهود المطلوب لإنجاز المهمة لم يتغير ولكن فترة الإنجاز هى التى تغيرت. وفى المقابل إذا كانت فترة إنجاز المهمة غير قابلة للتغيير، فيمكن لمدير المشروع أن يخصص الموارد المطلوبة لبذل المجهود المطلوب لإنجاز المهمة فى الوقت المحدد.

٤-٣ تقدير ميزانية المشروع Project Budget Estimation

تعتبر عملية تقدير ميزانية المشروع من الأنشطة المهمة والدرجة لمدير المشروع. هناك العديد من الأساليب المقترحة لتقدير التكاليف بعضها لا يعتمد على أساس علمي، والبعض الآخر يعتمد على خبرة القائمين على عملية التقدير استناداً إلى خبرة هؤلاء الأفراد فى مشاريع سابقة مشابهة فى تقدير ميزانية المشروع. وهناك طرق أخرى تسمى «التقدير بالقياس» وتقوم على القياس بمشاريع سابقة مشابهة لمجال المشروع الحالى نفسه. وهناك طرق تستند إلى أساس علمي تستخدم منها طريقتان على نطاق واسع هما:

١- تقدير مبنى على جدولة المشروع.

٢- تقدير مبنى على الخوارزميات.

وسوف نقوم الآن بعرضهما بشيء من التفصيل.

٤-٣-١ تقدير الميزانية حسب الجدولة Schedule-driven Budget Estimation

يعتمد هذا الأسلوب على وجود جدول لمهام المشروع مسبقاً وأن يكون قد تم توزيع الموارد المطلوبة على المهام. وتعرف هذه العملية بأنها تمثل الجدولة الأساسية أو المبدئية للمشروع وذلك لإمكانية التعديل فيها لاحقاً حسب التغيرات التى تطرأ على المشروع، مما يؤدي إلى وجود أكثر من جدول للمشروع. وبعد ذلك يتم تقدير تكلفة كل مهمة فى الجدول بدقة ويتم تجميعها ثم تضاف إليها التكاليف الثابتة للمشروع Fixed Costs. يتم إدخال تلك المعلومات من خلال استخدام إحدى أدوات إدارة المشاريع مثل Microsoft Project لحساب الميزانية المقترحة للمشروع. يجب الأخذ فى الاعتبار

أن تلك التقديرات تكون معرضة للتغيير في أثناء تنفيذ المشروع نظراً لإمكانية تغير تكلفة الموارد المطلوبة عند طلبها عن التكلفة المقدرة أثناء إعداد الخطة بشكل كبير نظراً لظروف غير متوقعة مثل الحروب أو الأزمات. يتم تقدير تكلفة الموارد البشرية عن طريق حساب معدل الأجر اليومي أو الأسبوعي أو الشهري أو خلال فترة محددة للأفراد المشاركين في عمليات التطوير، وكذلك حساب معدل الأجر الإضافي المحتسب لساعات العمل الإضافية التي يكلف بها بعض الأفراد لإنجاز مهامهم في وقت أقل من المقدر في الخطة. يتم تجميع كل هذه التكاليف ويضاف إليها التكاليف الثابتة لكل مهمة. يوضح شكل (٤-٦) مثلاً لاستخدام أداة Microsoft Project في حساب التكلفة الكلية لكل مهمة لأحد المشاريع.

Task Name	Duration	Start	Finish	Predecessors	Resource Name	Fixed Cost	Baseline Cost	Total Cost
1	مخطط نظام	10.5 days	Sat 12/05/07 Wed 23/05/07			0.000000 ر.س.	0.000000 ر.س.	27700
2	✓ تصميم المتطلبات	5 days	Sat 12/05/07 Wed 16/05/07		Mohsen	0.000000 ر.س.	0.000000 ر.س.	4000
3	✓ تحليل المتطلبات	5 days	Thu 17/05/07 Wed 23/05/07	2	Junaid, Sammak	0.000000 ر.س.	0.000000 ر.س.	16000
4	تحليل الجدول	3 days	Mon 21/05/07 Wed 23/05/07	FS+2 days	Mohsen, Sammak	0.000000 ر.س.	0.000000 ر.س.	7200
5	✓ تطوير الواجهات	0 days	Thu 24/05/07 Thu 24/05/07		Junaid, Mohsen	0.000000 ر.س.	0.000000 ر.س.	500
6	✓ تصميم نظام	4 days	Sat 26/05/07 Tue 29/05/07			0.000000 ر.س.	0.000000 ر.س.	14200
7	✓ تصميم المتطلبات	4 days	Sat 26/05/07 Tue 29/05/07	3	Sammak	0.000000 ر.س.	0.000000 ر.س.	6400
8	✓ تصميم قواعد البيانات	3 days	Sat 26/05/07 Mon 28/05/07	4	Junaid	0.000000 ر.س.	0.000000 ر.س.	4800
9	✓ تصميم المواجهات	3 days	Sat 26/05/07 Mon 28/05/07		Mohsen	0.000000 ر.س.	0.000000 ر.س.	3000
10	✓ اختبار النظام	15.5 days	Sat 26/05/07 Wed 13/06/07			0.000000 ر.س.	0.000000 ر.س.	52600
11	✓ برمجة التطبيقات	11.5 days	Sat 26/05/07 Sat 09/06/07	FS+1	Junaid, Khalid	0.000000 ر.س.	0.000000 ر.س.	27600
12	✓ اختبار التطبيقات	10 days	Sat 02/06/07 Wed 13/06/07		Sammak, Mohsen	0.000000 ر.س.	0.000000 ر.س.	24000
13	✓ إعداد قاعدة البيانات	0.000000	Sun 13/05/07 Sun 03/06/07			0.000000 ر.س.	0.000000 ر.س.	2800
14	✓ مراجعة الخطة ١	2 hrs	Sun 13/05/07 Sun 13/05/07		Sammak	0.000000 ر.س.	0.000000 ر.س.	400
15	✓ مراجعة الخطة ٢	2 hrs	Sun 20/05/07 Sun 20/05/07		Junaid	0.000000 ر.س.	0.000000 ر.س.	400
16	✓ مراجعة الخطة ٣	2 hrs	Sun 27/05/07 Sun 27/05/07		Sammak	0.000000 ر.س.	0.000000 ر.س.	400
17	✓ مراجعة الخطة ٤	2 hrs	Sun 03/06/07 Sun 03/06/07		Junaid	0.000000 ر.س.	0.000000 ر.س.	400

شكل (٤-٦) حساب تكلفة المشروع بناءً على جدول المشروع

نلاحظ من الشكل ما يلي:

- يتم إضافة التكلفة الثابتة لمعالم Milestones ومستلمات Deliverables كل مهمة.
- يتم إضافة التكلفة الثابتة لمهام المراجعة Recurring Tasks.
- تعتبر التكلفة الكلية هي تكلفة الخطة الأساسية Baseline للمشروع مضافاً إليها التكلفة الثابتة للمهام.

٤-٣-٢ تقدير الميزانية حسابياً Algorithmic Budget Estimation

مع كثرة الاختلافات التي حدثت في مشاريع كثيرة في السابق بين التكلفة المقدرة والتكلفة الفعلية للمشروع، أصبح لدى القائمين على إدارة المشروع قناعة كبيرة بضرورة أخذ طرق أخرى في الحسبان لتقدير تكلفة المشروع، إذ لا يجب الاعتماد على طريقة واحدة فحسب. ولذلك فإنه من الأفضل الاستعانة بالطرق الرياضية في تقدير التكلفة للتحقق من دقة الطرق الأخرى. تعتمد الطرق الحسابية على بعض الصيغ الرياضية لتقدير تكلفة الموارد البشرية على وجه الخصوص من حيث حساب تكلفة المجهود المبذول وحجم المشروع. يتم تمثيل حجم المشروع من خلال عدة عناصر مثل:

أ- عدد أسطر الأكواد (LOC) Lines of Code

يصف عدد أسطر أكواد البرامج المكتوبة بدون الأخذ في الاعتبار الأسطر التوضيحية Comment lines. وتحسب الأسطر دائماً بالآلاف وتأخذ الرمز KLOC في الصيغ الرياضية.

ب- نقاط الوظيفة (FPs) Function Points

تعتبر أحد مقاييس حساب حجم المشروع ولكن في مستوى أعلى من مستوى عدد الأسطر. يتم حسابها بناءً على توصيف المنتج البرمجي من خلال وظائف أساسية، هي:

- المدخلات Inputs.

- المخرجات Outputs.

- ملفات البيانات Data Files.

- الاستفسارات Inquiries.

- الواجهات الخارجية External Interfaces.

يتم حساب عدد النقاط حسب تلك الوظائف مع الأخذ في الاعتبار معاملات وزنية Weighting factors لضبط عملية حساب التكلفة حسب درجة تعقيد الوظيفة.

ج - نقاط الكائن (Object Points (Ops؛

تمثل مقياساً آخر في مستوى أعلى لحساب حجم المشروع. وهي أيضاً تعتمد على توصيف المنتج البرمجي من خلال مجموعة من الخصائص المقدمة مثل:

- المكونات البرمجية Software Components.

- التقارير Reports.

- واجهات المستخدم User Interfaces.

يتم حساب مجموع النقاط للكائنات مع الأخذ في الاعتبار معاملات وزنية لضبط عملية حساب التكلفة حسب درجة تعقيد كل كائن.

يتم استخدام هذه المعلومات عن حجم المشروع في صياغة النموذج الرياضي المستخدم في تقدير تكلفة المشروع ويأخذ الشكل العام:

$$\text{Effort} = C * \text{Size}^K$$

حيث تأخذ المعاملات C, K قيمةً تجريبية مأخوذة من مشاريع سابقة. يمثل المعامل K درجة التعقيد بالمشروع وتأخذ القيم من ١ إلى ١,٥، ويمثل المعامل C درجة الصعوبة في المشروع التي يتم تقييمها من خلال عدة خصائص مثل مهارات فريق العمل ودرجة إتاحة الموارد المطلوبة ودرجة المرونة المطلوبة للمنتج البرمجي وغيرها.

سنقدم الآن عرضاً لأحد أشهر الطرق الرياضية المستخدمة في تقدير تكلفة المشروع وهي طريقة COCOMO.

طريقة COCOMO؛

يهدف النموذج الرياضي Constructive Cost Model إلى حساب تكلفة المجهود المبذول لإنجاز مهام المشروع. لقد تم تقديم أول نسخة منه في عام ١٩٨١ باسم COCOMO 81، وتم تطويرها فيما بعد لتظهر النسخة المطورة عام ٢٠٠٠ باسم COCOMO II وسوف نركز الآن على النسخة المطورة للأسباب التالية: (Pressman, 2001).

- تناسب أساليب التطوير الحديثة مثل أسلوب التكرار مع الزيادة وأسلوب البرمجة الموجهة بالكائنات Object-Oriented Programming.

- تناسب أيضاً استخدام أساليب النمذجة Prototyping واستخدام مولدات البرامج والتطوير باستخدام المكونات Component-Based Development.

- تلائم التطبيقات الكبيرة والمعقدة.

ومن السمات المهمة لطريقة COCOMO II أنها تعتمد على مرحلة التطوير التي يتم التقدير خلالها، مما يؤدي إلى دقة عملية التقدير للتكلفة. ولهذا فإن طريقة COCOMO II تقدم ثلاثة نماذج مختلفة تقابل مراحل التطوير الأولى للمشروع هي:

١- نموذج تكوين التطبيق Application Composition Model

يستخدم في بداية عملية تحليل المتطلبات، ويستخدم نقاط الكائن الموزونة Weighted Object Points (Ops) لتحديد حجم المهمة المطلوب تقديرها. يتم حساب نقاط الكائن لكل من واجهات المستخدم وتقارير المنتج البرمجي ومكونات المنتج البرمجي. وبعد ذلك يتم تخفيض عدد النقاط حسب نسبة الكائنات التي تم الحصول عليها باستخدام أسلوب إعادة الاستخدام Reuse Objects واحتساب نقاط الكائنات الجديدة فقط. يتم صياغة معادلة حساب نقاط الكائن (NOP) كما يلي:

$$NOP = OP * [(100 - \% Reuse)] / 100$$

يستخدم النموذج المقترح عاملين أساسيين لتحديد مستوى إنتاجية عملية التطوير Development Productivity Level وهما عامل الخبرة والمقدرة لفريق التطوير وعامل مكانة ومقدرة المؤسسة. يوضح شكل (٤-٧) المستويات الخمسة لإنتاجية عملية التطوير.

وبعد ذلك يمكننا حساب قيمة المجهود المطلوب من فريق العمل كما يلي:

$$Effort = NOP / Productivity$$

فمثلاً إذا كان عدد نقاط الكائن هي (٥٠٠) والإنتاجية هي (٤)، فإن قيمة المجهود المطلوب هي (١٢٥) شخص/شهر، وإذا كانت إنتاجية فريق العمل هي (٥٠) فإن المجهود المطلوب يصبح (١٠) أشخاص/شهر فقط.

الإنتاجية NOP/month	خبرة ومقدرة المطورين ونضج ومقدرة المؤسسة
٤	قليلة جداً
٧	قليلة
١٣	عادية
٢٥	عالية
٥٠	عالية جداً

شكل (٤-٧) قياس الإنتاجية في طريقة COCOMO II

٢- نموذج التصميم المبكر Early Design Model

يستخدم بعد اكتمال عملية تحليل المتطلبات ويستخدم نقاط الوظيفة (FPs) لحساب حجم المهمة المطلوب تقديرها. تقدم طريقة COCOMO II جدولاً قياسياً لتحويل عدد نقاط الوظيفة إلى عدد أسطر الأكواد (KLOCs) الذي يستخدم بدوره في تقدير قيمة المجهود المطلوب كما يلي:

$$\text{Effort} = C * \text{SizeK} * m + \text{autoeffort}$$

يتم تحديد قيم المعاملات المذكورة في الصيغة بناءً على معلومات من (٨٣) مشروعا سابقاً كما يلي:

- يأخذ المعامل C قيمة (٢,٥) في مرحلة التصميم الأولية.
- يأخذ المعامل K قيمةً تراوح بين (١,٠١) إلى (١,٢٦).
- يتم تحديد قيم المعامل m من خلال سبع خصائص هي:
- درجة الثقة في المنتج البرمجي Reliability.
- درجة إعادة الاستخدام Reusability.
- درجة صعوبة بيئة التشغيل Platform.
- مقدرة الأشخاص Capability.
- خبرة الأشخاص Experience.
- الجدول Schedule.
- التسهيلات المقدمة Facilities.

تأخذ كل خاصية قيمة معينة، وإذا كانت الخاصية غير مؤثرة فإنها تأخذ القيمة (١). تكون قيمة المعامل m هي حاصل ضرب القيم السبع.

يعنى المعامل autoeffort بالمجهود الذي يقوم به المطورون في أثناء عملية توليد الأكواد آلياً وكذلك عملية تكاملها مع الأكواد الأخرى التي ينتجها المطورون أنفسهم.

٣- نموذج ما بعد التصميم المعماري Post-Architectural Model

يستخدم بعد تحديد التصميم المعماري للبرمجيات ويستخدم عدد سطر الأكواد (LOCs) لتحديد حجم المهمة المطلوب تقديرها. ويستخدم الصيغة الرياضية المذكورة نفسها في النموذج السابق (نموذج التصميم المبكر) ولكن يختلف عنه في تفاصيل

المعاملات. يستخدم هذا الأسلوب (١٧) خاصية موزعة على أربع مجموعات. تأخذ الخواص غير المؤثرة القيمة الأحادية، ويأخذ المعامل m حاصل ضرب قيم الخواص السبع عشرة. يوضح شكل (٤-٨) مجموعة الخواص التي تقود عملية تقدير التكلفة.

اسم الخاصية	كود الخاصية	تصنيف الخاصية
درجة الثقة المطلوبة للنظام	RELY	خصائص المنتج
درجة تعقيد وحدات النظام	CPLX	
درجة التوثيق المطلوب	DOCU	
حجم قواعد البيانات	DATA	
النسبة المئوية المطلوبة لإعادة استخدام المكونات	RUSE	
قيود وقت التنفيذ	TIME	خصائص الحاسب
فترة بقاء النظام الأساسى للتطوير	PVOL	
قيود التخزين	STOR	
القدرة على تحليل النظام	ACAP	خصائص شخصية
التواصل الشخصى	PCON	
خبرة المبرمجين فى مجال المشروع	PEXP	
قدرة المبرمجين	PCAP	
خبرة المحللين فى مجال المشروع	AEXP	
الخبرة فى استخدام أدوات التطوير	LTEX	
مدى استخدام أدوات البرمجيات	TOOL	خصائص المشروع
ضغط الجدولة	SCED	
إمكانية العمل على عدة مواقع على الويب	SITE	

شكل (٤-٨) الخواص المؤثرة فى تحديد قيمة المجهود

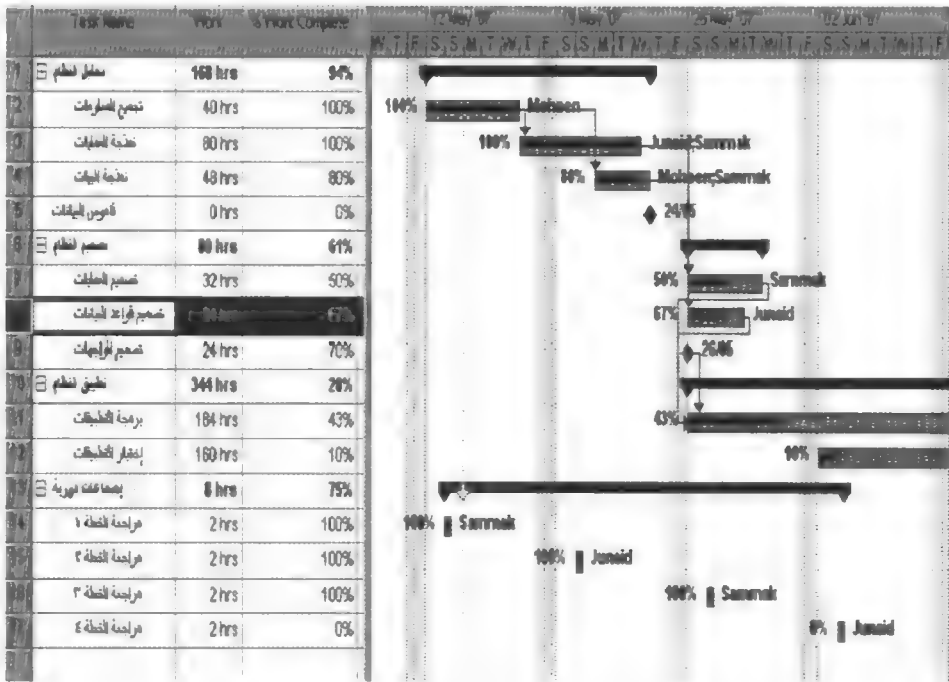
٤-٤ متابعة تقدم المشروع Tracking Project Progress:

يبدأ مدير المشروع فى متابعة تنفيذ مهام المشروع ومقارنة المنجزات مع المعلومات الموجودة بجدولة المشروع وكذلك مقارنة التكلفة الفعلية مع التكلفة المقدرة لكل مهمة. فمن الطبيعى أن تحدث عدة تغييرات فى كل من الجدولة وتكلفة الموارد المطلوبة وهذا يتطلب متابعة جيدة لإمكانية التحكم فى التغيير بصورة أفضل وبأقل تأثير على وقت تسليم المشروع وتكلفته المقدرة. وأحيانا تكون المعلومات المجمعة من متابعة الجدولة متعارضة مع المعلومات المجمعة من متابعة التكلفة.

حيث يكون إنجاز مهام المشروع متوافقاً مع الجدولة في خطة المشروع ولكن مع تجاوز في الميزانية المقدرة، وأحياناً يكون المشروع متوافقاً مع الميزانية المقدرة ويكون متأخراً عن جدولة المشروع. والآن سنعرض كلا المسارين بشيء من التفصيل.

٤-٤-١ متابعة جدول المشروع Tracking the Project Schedule

يتم استخدام مخطط جانت GANTT chart لتسجيل معلومات الأجزاء المنفذة في كل مهمة وكذلك تسجيل أوقات بداية ونهاية التنفيذ الفعلي لكل مهمة والأفراد القائمين على إنجازها ونسبة مشاركة كل فرد في المهمة. تمثل هذه المعلومات الجدولة الفعلية Actual Schedule للمشروع مقابل الجدولة الأولية Baseline Schedule. يوضح شكل (٩-٤) كيفية تسجيل معلومات المتابعة باستخدام مخطط جانت، إذ يتم تظليل المهمة المنجزة بلون داكن حسب درجة اكتمال المهمة (تمت، لم تبدأ بعد، تم إنجاز جزء منها). ويتم أيضاً تسجيل معلومات عن حجم الإنجاز مقارنة بالموارد المستخدمة في كل مهمة.



شكل (٩-٤) استخدام مخطط جانت لمتابعة جدولة المشروع

٤-٤-٢ متابعة الميزانية Tracking The Budget

تعتبر عملية متابعة الميزانية هى عملية مكاملة لعملية متابعة الجدولة. فمن الممكن أن تتم متابعة التكلفة الفعلية للمهام المنجزة أو التى مازالت فى طور التنفيذ. بعد ذلك يتم تلخيص تلك المعلومات المجموعة وصياغتها فى شكل تقرير متابعة الميزانية ويتم تقديمه للإدارة العليا لاتخاذ القرار المناسب. يوضح شكل (٤-١٠) كيفية حساب التكلفة المبدئية Baseline Cost، والتكلفة الفعلية Actual Cost، وكذلك التكلفة المتبقية Remaining Cost لكل مهمة.

Task Name	Duration	Start	Finish	Resource Names	Planned Cost	Baseline Cost	Actual Cost	Remaining Cost
1 تطوير النظام	10.5 days	Sat 12/05/07	Wed 23/05/07	Sarwanik,Mohsen	٥٠٠,٠٠٠ ر.س.	٢١,٠٠٠,٠٠٠ ر.س.	١٩,٢٩٠,٠٠٠ ر.س.	٢٢,١١٠,٠٠٠ ر.س.
5 قسوس البيانات	0 days	Thu 24/05/07	Thu 24/05/07	Junaid,Mohsen	٥٠٠,٠٠٠ ر.س.	٠ ر.س.	٠ ر.س.	٥٠٠,٠٠٠ ر.س.
6 تصميم النظام	4 days	Sat 26/05/07	Tue 29/05/07	Mohsen,Junaid,S	١٠٠,٠٠٠ ر.س.	١٢,١٠٠,٠٠٠ ر.س.	١١,٤١١,٠٠٠ ر.س.	١٢,٦٨٩,٠٠٠ ر.س.
10 تطوير النظام	15.5 days	Sat 26/05/07	Wed 13/06/07	Sarwanik,Mohsen	١,٠٠٠,٠٠٠ ر.س.	٥١,١٠٠,٠٠٠ ر.س.	٢٩,١٤٤,٠٠٠ ر.س.	١٩,٩٥٦,٠٠٠ ر.س.
13 بصماتنا لبرية	16.75 days	Sun 13/06/07	Sun 13/06/07	Sarwanik,Junaid	١,٠٠٠,٠٠٠ ر.س.	١٤,٠٠٠,٠٠٠ ر.س.	١٤,٠٠٠,٠٠٠ ر.س.	٥٢,٥٠٠,٠٠٠ ر.س.

شكل (٤-١٠) حساب التكلفة المبدئية والفعلية لمهام المشروع

تحليل القيمة المكتسبة Earned Value Analysis

يستخدم أسلوب القيمة المكتسبة لتحديد مدى حيد المشروع عن الميزانية المقدرة له فى خطة المشروع (Pressman, 2001). يستخدم هذا الأسلوب التقديرات الأولية فى الخطة المبدئية للمشروع والتكلفة الفعلية حتى الآن لتحديد الوضع المالى الحالى للمشروع. يتم تطبيق هذا الأسلوب على كل مهمة على حدة وعلى المشروع كله. يحتوى أسلوب تحليل القيمة المكتسبة على مجموعة المقاييس التالية (شكل (٤-١١)):

التكلفة المقدرة للأعمال المجدولة BCWS

تمثل القيمة المكتسبة للمهمة التى من المفترض أن تستخدم الموارد المخصصة لها فى جدول المشروع، وأن يتم إنجازها بالفعل قبل تاريخ التقييم، وتأخذ المهام التى لم تبدأ بعد القيمة المكتسبة صفراً.

تكلفة الأعمال المنجزة حسب الجدولة BCWP،

تمثل القيمة المكتسبة للمهمة بناءً على نسبة الإنجاز المقدرة بالجدول التي تمت بالفعل قبل تاريخ التقييم.

التكلفة الفعلية للأعمال المنجزة ACWP،

تمثل مجموع التكلفة الفعلية للمهام المنجزة والتكاليف الثابتة وأى تكاليف أخرى يتم إدخالها يدوياً.

- التباين المجدول SV،

يمثل الفرق بين تكلفة الأعمال المنجزة حسب الجدولة والتكلفة المقدرة فى الجدول ($BCWP - BCWS$)، وإذا كان قيمة SV تساوى الصفر، فهذا يعنى أن المهمة قد تم إنجازها بالكامل أو أنها لم تبدأ بعد، وتأخذ قيمة سالبة عندما تكون المهمة غير مكتملة.

- تباين التكلفة CV،

يمثل الفرق بين تكلفة الأعمال المنجزة حسب الجدولة والتكلفة الفعلية للأعمال المنجزة ($BCWP - ACWP$). وهو مقياس يشير إلى مدى زيادة أو نقصان التكلفة الفعلية للمهمة المنجزة عن التكلفة المقدرة بالخطة المبدئية للمشروع.

- التقدير عند الاكتمال EAC،

يمثل التكلفة المقدرة للمهمة عند اكتمالها بناءً على التقدم الحالى فى إنجاز المهمة. وهو يساوى قيمة ACWP للمهام المكتملة بالفعل.

- الميزانية عند الاكتمال BAC،

يمثل التكلفة المجدولة للمهمة عند اكتمالها بناءً على التقدم الحالى فى إنجاز المهمة. وهو يساوى قيمة BCWP للمهام المكتملة بالفعل.

- التباين عند الاكتمال VAC،

يمثل الفرق بين الميزانية عند الاكتمال والتقدير عند الاكتمال (BAC - EAC). وهو يساوى قيمة تباين التكلفة CV للمهام المكتملة، ويساوى الصفر للمهام التي لم تبدأ بعد، وتكون قيمته سالبة عند تجاوز التكلفة الفعلية عن التكلفة المجدولة للمهمة، وتكون قيمته موجبة عندما تكون التكلفة الفعلية أقل من التكلفة المجدولة للمهمة.

Task Name	BOS	BOP	ACWP	SV	CV	EAC	BAC	VAC
1 تصميم نظام	1,000,000 ر.س.	1,000,000 ر.س.	1,100,000 ر.س.	0 ر.س.	-100,000 ر.س.	2,000,000 ر.س.	2,000,000 ر.س.	-1,000,000 ر.س.
5 تكوين البيانات	0 ر.س.	0 ر.س.	0 ر.س.	0 ر.س.	0 ر.س.	500,000 ر.س.	500,000 ر.س.	0 ر.س.
6 تصميم نظام	0 ر.س.	0 ر.س.	0 ر.س.	0 ر.س.	0 ر.س.	1,500,000 ر.س.	1,500,000 ر.س.	0 ر.س.
10 تطبيق نظام	0 ر.س.	0 ر.س.	0 ر.س.	0 ر.س.	0 ر.س.	1,000,000 ر.س.	1,000,000 ر.س.	0 ر.س.
13 بصمات اختبارية	0 ر.س.	0 ر.س.	1,000,000 ر.س.	0 ر.س.	-1,000,000 ر.س.	2,000,000 ر.س.	2,000,000 ر.س.	-1,000,000 ر.س.

شكل (١١-٤) تحليل القيمة المكتسبة

٤-٥ إدارة الموارد البشرية Human Resources Management:

تركز عملية إدارة الموارد البشرية على أنشطة اكتساب الأفراد وتحفيزهم بالإضافة إلى عمليات تنظيم وإدارة الاتصال بينهم في المشروع وكذلك تنمية قدراتهم وتدريبهم. تتسم عملية إدارة الموارد البشرية بالديناميكية نظرا للتغيير المستمر في بيئة نظام الأعمال وهو تغيير يؤثر في هيكل المؤسسة مما يتطلب مهارات إبداعية من مدير المشروع لإعادة هيكلة الموارد البشرية. سوف نستعرض الآن أهم أنشطة إدارة الموارد البشرية.

٤-٥-١ اكتساب وتحفيز الأفراد Acquiring and Motivating People:

يتم تشكيل فريق العمل بالمشروع من واقع خطة المشروع بمساعدة خطة إدارة الأفراد المصاحبة لها. وتعتبر عملية تحفيز واكتساب الأفراد هي جوهر عملية بناء أو تكوين فريق العمل. تتضمن عملية تكوين فريق العمل اختيار مدير الفريق وأعضاء الفريق حسب المهارات المطلوبة لإنجاز الأعمال بالمشروع. وتعتبر عملية اختيار مدير المشروع من العوامل المؤثرة جداً في نجاح المشروع. يجب أن تتوافر مجموعة من الخصائص المهمة في شخصية مدير المشروع مثل:

- المقدرة على قيادة فريق العمل وتحقيق الأهداف المطلوبة.
 - المقدرة على إقناع فريق العمل برؤيته.
 - المهارات الأساسية لأنشطة إدارة المشروع مثل التخطيط والجدولة والمتابعة وتقدير التكلفة وغيرها.
 - مهارة اكتساب الأفراد وتحفيزهم.
 - مهارة التنسيق والاتصال بين أفراد الفريق.
 - المرونة في التفكير ومهارة التفكير الاستدلالي.
 - معرفة جيدة بنظم الأعمال.
 - الثبات عند مواجهة الأزمات.
- يقوم مدير المشروع بتنفيذ مجموعة من المهام لتكوين فريق عمل فعال يناسب مشروع الأعمال الحالي، وتتضمن المهام ما يلي:
- تشكيل فريق العمل.
 - تحفيز أفراد الفريق.
 - تنمية قدرات أفراد الفريق.
 - تنسيق الاتصال بين أفراد الفريق وبين فرق العمل.

٤-٥-٢ تكوين فريق العمل Team Work Formation:

تحتاج مشاريع البرمجيات الكبيرة المعقدة إلى العديد من الخبرات في مجالات مختلفة مثل خبراء في مجالات تقنية المعلومات مثل التحليل والتصميم والبرمجة

والاختبار والتوثيق، بالإضافة إلى خبراء في إدارة المشاريع، وخبراء في عمليات دعم العميل، وغيرها من الخبرات. وبالطبع فإن مدير المشروع يأمل أن يضم أفضل الخبراء في كل مجال إلى فريق العمل الذي يحتاج إليه في المشروع، غير أن ذلك الأسلوب لا يضمن بالقطع نجاح المشروع إلا إذا كانوا يمتلكون مهارات شخصية (بالإضافة إلى مهاراتهم الفنية العالية) تمكنهم من العمل معاً بروح الفريق الذي يتطلب قدراً كبيراً من التعاون والمشاركة والمرونة والتفاهم بينهم. ويعتبر ذلك من أهم أدوار مدير الفريق ذلك يشمل الواجبات التالية:

- تحديد المهام المطلوبة بدقة.
- تحديد الخبرات المطلوبة لإنجاز المهام.
- ترشيح عدد من الأفراد ذوي الخبرات المطلوبة.
- اختيار أفراد الفريق المتجانسين والأقدر على التفاهم والعمل معاً بتعاون تام.
- المتابعة والتوجيه المستمر لفريق العمل لضمان تحقيق الأهداف المطلوبة.
- التنسيق بين أفراد الفريق.
- حل التعارض الذي ينشأ بين أفراد الفريق في أسرع وقت.
- وفيما يلي عرض لبعض المهارات الفنية المطلوبة لمشاريع البرمجيات.
- تحليل وتصميم النظم.
- تصميم قواعد البيانات.
- برمجة التطبيقات.
- اختبار البرمجيات.
- توثيق التطبيقات.
- تصميم شبكات الحاسب.
- التصميم المعماري للتطبيقات.
- تثبيت واستخدام أدوات البرمجيات.
- الدعم الفني للعميل.

- وذلك بالإضافة إلى مجموعة من المهارات الشخصية لأفراد الفريق مثل:
- التحفيز الذاتى والطموح الشخصى.
- التواصل الجيد مع الآخرين.
- القدرة على التركيز فى التفاصيل.
- المرونة والنضج فى التفكير.
- الإحساس بالمسؤولية عند الطوارئ.

٤-٥-٣ اتصالات المشروع Project Communications:

المقصود بالاتصالات هو تبادل المعلومات داخل المشروع، أى أنها تتضمن أنشطة إرسال واستقبال الرسائل بين المشاركين فى المشروع. تعتبر هذه العملية ذات أهمية كبيرة لنظم مشاريع البرمجيات وتحتاج إلى إدارة جيدة لضمان الاستفادة من تداول المعلومات داخل المشروع. تركز عملية إدارة الاتصالات داخل المشروع على عدة مفاهيم أو وجهات نظر مثل أشكال الاتصالات وخطوط الاتصال وعوامل الاتصال والاتصال لحل التعارضات، وسوف نستعرضها الآن بإيجاز.

أشكال الاتصالات Forms of Communications:

يتم تبادل الرسائل بعدة أشكال مختلفة مثل:

- شفهيًا.
- كتابيًا.
- باستخدام الصور.
- باستخدام الرموز.
- يمكن أن تكون الرسالة سرية أو عامة وتأخذ شكلاً محدداً أو غير محددة الشكل، ويمكن أن تكون داخلية أو خارجية. تأخذ الاتصالات داخل المشروع عدة أشكال منها:
- اتصال عاды من شخص إلى شخص.
- اتصال بين الأفراد داخل غرفة الاجتماعات.

- محادثة هاتفية بين شخص وآخر.
- اتصال بين شخص وآخر عبر الإنترنت.
- اتصال عبر الفاكس.
- بريد عادى.
- اتصال عبر بريد إلكترونى.
- اتصال عبر صفحات الويب.

يتميز كل شكل من هذه الأشكال بمميزات خاصة تناسب ظروفًا معينة لإرسال واستقبال الرسائل. وفى الحقيقة يتم استخدام معظم هذه الأشكال فى أثناء إجراء الاتصالات داخل أى مشروع، مع الأخذ فى الاعتبار أن أكثر هذه الأشكال استخداماً هو أسلوب الرسائل الإلكترونية.

خطوط الاتصال Lines of communications:

تهتم عملية إدارة الاتصالات بالمشروع بتقليل خطوط الاتصال بين الأفراد وبين فريق العمل بقدر الإمكان. فمن الطبيعى أن تؤدي كثرة خطوط الاتصال إلى صعوبة فى عملية إدارة الاتصالات. تتطلب عملية إدارة الاتصالات تنظيم هيكل الاتصالات من خلال تنظيم الأفراد فى شكل فرق عمل وتحديد أسلوب الاتصال بينهم فى شكل هرمى. يتضمن الشكل الهرمى للاتصالات عدة خطوط اتصال فى الاتجاهين الرأسى والأفقى. وفى الاتجاه الأفقى توجد خطوط اتصال بين الأفراد، ويعتمد عدد الخطوط على عدد الأفراد فى هذا المستوى، إذ تزداد عدد خطوط الاتصال بشكل كبير جداً مع زيادة عدد الأفراد. ولتقليل عدد خطوط الاتصال يتم تجميع الأفراد فى شكل مجموعات عمل جزئية وتقتصر الاتصالات بينهم على الاتصال بين منسقى الفرق.

عوامل الاتصال Communication Factors:

من المعروف أن كثرة المعلومات عن اللازم أو قلتها عن الضرورى تؤدي إلى مشاكل كبيرة داخل المشروع. فيجب إمداد فرق العمل بالمعلومات المتعلقة بالمهام التى يقومون بإنجازها. ويجب إمداد باقى المشاركين فى المشروع بمعلومات عامة عن المنتجات والاتجاهات الخاصة بالمشروع حتى يكون لديهم فكرة واضحة عن

الوضع الحالي للمشروع. وهناك عامل آخر يخص الاتصال الأفقى بين الأفراد أو مجموعات العمل وهو التأكد من وصول المعلومات بشكل دقيق، ويتطلب ذلك الاهتمام بعملية اختيار الأفراد المسؤولين عن توصيل المعلومات. وينطبق المفهوم نفسه على الاتصالات التى تتم خلال اجتماعات العمل. فهناك اجتماعات عمل دورية خاصة بفريق العمل للتشاور وتبادل الآراء والمعلومات. وهناك اجتماعات عامة تشمل جميع المشاركين فى المشروع وتتضمن معلومات عامة عن مشروع الأعمال والمنجزات الحالية.

الاتصال لحل التعارضات Communication for Conflict Resolution

من الطبيعى أن يكون هناك كثير من التعارضات داخل المشروع يتم التغلب عليها غالباً من خلال الاتصال الجيد بين الأفراد أو فرق العمل. هناك عدة طرق تستخدم للتغلب على التعارضات الموجودة بالمشروع عن طريق الاتصال مثل:

الإجبار Forcing

يتم إجبار أحد فرق العمل المتضاربة على تنفيذ أحد الحلول وذلك على مسؤولية مدير المشروع؛ إذ إنه هو الذى اتخذ القرار باختيار هذا الحل. وهنا يجب على الفريق المتعارض أن يلتزم بتنفيذ ما تم إقراره من مدير المشروع.

التسوية Compromise

هو أسلوب يتطلب الاتصال وجهاً لوجه بين المتعارضين ومحاولة الوصول إلى حل وسط يرضى الطرفين.

المواجهة Confrontation

هى أكثر الطرق فعالية فى حل التعارضات، إذ يتم عرض الحقائق ومناقشة التبريرات الخاصة بكل مقترح، من ثم يتم غالباً الوصول إلى حل للمشكلة والاتفاق عليه.

التهذية Smoothing

هو أسلوب مؤقت لحل المشكلة وذلك بتقليل أهمية الأجزاء التى تحتوى على تعارضات والتركيز على الأجزاء التى لا تحتوى على تعارضات. أحياناً تتلاشى هذه

التعارضات مع التقدم فى إنجاز المهام بالمشروع. وأحياناً أخرى تعود تلك التعارضات بشدة أكبر.

الانسحاب Withdrawal،

يتم فى هذا الأسلوب حل مشكلة التعارضات بالانسحاب أحد الفرق المتعارضة من مناقشة موضوع التعارض مرة أخرى. ويعتبر هذا الأسلوب هو أسوأ الحلول لكونه يؤدى إلى تقليل خطوط الاتصال الضرورية بين فرق العمل بشكل خطير يؤثر سلباً فى نجاح المشروع.

٤-٥-٤ تنمية فريق العمل Team Work Development،

لا ينبغي ترك الفريق، بعد تشكيكه، منعزلاً فى أثناء العمل، ولكن يجب على مدير الفريق أن يقوم بمتابعة أفراد الفريق فى أثناء تأدية مهامه ويتدخل فى الوقت المناسب بإجراء تعديلات ضرورية على نشاطات المشروع لضمان نجاحه. نعرض الآن لبعض النشاطات التى يقوم بها مدير المشروع لمساعدة الفريق فى إنجاز أعمالهم بنجاح:

- التأكد من توافر القدر الكافى من الاتصال بين أفراد الفريق.
- التأكد من أن أفراد الفريق يتعاملون معاً باحترام.
- التأكد من أن كل فرد يتفهم دوره وواجباته بوضوح تام.
- التأكد من فهم كل أفراد الفريق لأهداف المشروع ودعمها.
- التأكد من توافر القدر الكافى من الاتفاق بين أفراد الفريق على العمليات المنجزة.
- التأكد من أن أفراد الفريق ليسوا من مدمنى التقاعس.

يتميز فريق العمل المتجانس بالقدرة على تجاوز الخلافات بينهم بأنفسهم وحل التعارض فى وجهات النظر أو الفروق فى المستويات بينهم، وبعد فترة يصبحون أكثر ترابطاً وتفهماً مما يؤدى إلى زيادة ملحوظة فى كفاءتهم وإنتاجهم. ويجب على مدير الفريق أن يعطى أعضاء الفريق الفرصة والوقت لزيادة الارتباط والتفاهم بينهم ويتدخل فقط فى المشكلات الكبيرة أو التى تنتج من خارج الفريق. ويمكن أيضاً لمدير الفريق أن يقوم بعمل لقاءات عمل دورية لتبادل الأفكار ودراسة المشكلات وتقريب وجهات النظر للوصول إلى أفضل الحلول. كل ذلك يعود بالنفع على أفراد الفريق لتنمية مهاراتهم وعلى المشروع فى النهاية.

٤-٦ إدارة المخاطر Risk Management:

من النادر أن يخلو أى مشروع برمجيات من تعرضه لنوع من المخاطر. فهناك دائماً أسباب تؤدي إلى حدوث المخاطر مثل عدم الوصف الدقيق لأى جزء من أجزاء المشروع أو عدم الفهم الجيد له أو عدم دراسته جيداً أو عدم توثيقه جيداً أو ضعف الاتصال بين أفراد الفريق، وغيرها من الأسباب. فمن المعروف أن عملية تخطيط المشروع تتضمن العديد من الأنشطة بدءاً من تحديد المتطلبات والمستلزمات، مروراً بتحليل المهام وجدولتها، وتحديد الموارد وتقديرها، وكذلك تحديد العمليات والمنهجيات والأدوات المطلوبة لإنجاز المهام. فمن الصعوبة أن يتم تنفيذ كل هذه الأنشطة بطريقة مثالية بدون أى مشكلات أو مخاطر. بالإضافة إلى أن هذه القائمة الطويلة من الأنشطة فى حد ذاتها تحمل مخاطر عديدة. وعلى هذا فمن الطبيعى أن يتم إجراء تعديلات على خطة المشروع وهو فى طور التنفيذ.

المخاطرة Risk هى مشكلة ناتجة من عدم التأكد من عملية أو أسلوب أو موارد يتم استخدامها فى المشروع ولها تأثير سلبى فى المشروع، ويمكن التغلب عليها عند حدوثها باعتماد خطة خاصة بالمخاطر المتوقعة. ومثال ذلك، عندما يقرر مدير المشروع استخدام تقنية جديدة فى المشروع تكون نتائجها غير مؤكدة وجديدة على فريق العمل، أو استخدام موارد جديدة، أو انضمام عضو جديد لفريق العمل أو قلة الاتصال والتفاهم بين أعضاء الفريق، كل ذلك يؤدى إلى نوع من المخاطر من المحتمل أن تؤثر سلباً فى المشروع.

٤-٦-١ تحديد المخاطر Risk Identification:

كل مشروع يواجه مجموعة من المخاطر المحتملة حسب طبيعته، وعلى مدير المشروع أن يكتشف أو يتوقع كل مواطن المخاطر الممكنة وأن يتعامل معها بجدية وصدق وعدم التهور من شأنها. ويمكن لمدير المشروع أن يستعين بخبرات مديرين أكثر خبرةً منه فى نظم أعمال مشابهة. وبعد تحديد نوع الخطر، يجب وصفه بوضوح لفريق العمل ووصف تأثيره فى المشروع والأفعال المطلوبة للتغلب عليه. ويعتبر ضعف المفهوم فى أى مرحلة من مراحل إدارة وتنفيذ المشروع هو المصدر الرئيسى للمخاطر فى مشاريع البرمجيات. وفيما يلى عرض لأهم المصادر الرئيسية لحدوث المخاطر فى مشاريع البرمجيات:

- التفاوض الزائد حول استخدام تقنيات جديدة.
 - عدم فهم التأثيرات الحقيقية المؤثرة لبعض التقنيات.
 - عدم الفهم التام لمتطلبات العميل.
 - التغييرات المستمرة فى متطلبات العميل.
 - عدم كفاية الموارد المطلوبة للمشروع.
 - قلة خبرة أعضاء فريق العمل.
 - التوقعات غير الحقيقية حول مهارة وإنتاجية بعض الموارد البشرية.
 - تجاهل وجهات نظر وطلبات المستخدمين للمشروع.
 - عدم وضوح ودقة المتطلبات الوظيفية لنظام الأعمال.
 - التقييم غير الدقيق لموارد المشروع.
 - إعداد خطة المشروع بطريقة متعجلة غير دقيقة.
 - عدم إشراك باقى المساهمين فى المشروع فى أثناء إعداد خطة المشروع.
- يجب على مدير المشروع أن يقوم بتحديد قائمة بالمخاطر المتوقع حدوثها فى أثناء تنفيذ المشروع. تعتبر هذه القائمة غير نهائية ولكن يتم تحديثها باستمرار مع التقدم فى إنجاز مهام المشروع. ولتجهيز تلك القائمة يقوم مدير المشروع بالتركيز على الموضوعات الرئيسية التى تضمنها خطة المشروع وهى:
- مواصفات المنتجات والمستلزمات للمشروع.
 - قائمة المهام وجدولتها.
 - الأهداف وطريقة قياسها.
 - الموارد البشرية.
 - العمليات والمنهجيات.
 - الأدوات والمعدات.
- فيقوم مدير المشروع بدراسة كل بند من البنود السابقة بعناية مع أكبر عدد ممكن من المشاركين فى المشروع وتبادل الآراء ووجهات النظر للوصول إلى القائمة المبدئية للمخاطر المتوقعة.

٤-٦-٢ تقييم المخاطر Risk Assessment

يتم تقدير درجة المخاطر المتوقعة باستخدام طريقتين:

الأولى: احتمالية حدوث المخاطر Probability:

وهي تمثل مقياساً مدرجاً يبدأ من (١٠٪) إلى (٩٠٪) وإذا وصلت النسبة إلى (١٠٠٪) فإنها لا تعتبر مخاطرة إنما تصبح قيداً مؤكداً في المشروع.

الثانية: التأثير السلبي في المشروع Negative Impact:

وهو مقياس يتكون من خمس درجات هي:

- هائل/كارثي Catastrophic.

- حرج Critical.

- خطير Serious.

- محتمل Tolerable.

- هامشي Marginal.

يتم تمثيل قيمة التأثير السلبي في المشروع من خلال تقدير قيمة تكلفة التغلب عليه، وبعد ذلك يتم حساب قيمة التعرض للمخاطرة لكل نوع من المخاطر المتوقعة كما يلي:

قيمة المخاطرة = احتمالية المخاطرة × قيمة التأثير السلبي

٤-٦-٣ معالجة المخاطر Risk Handling

بعد عمليات تحديد المخاطر وتقييمها، ننتقل الآن إلى نشاط مهم جداً وهو التخطيط لتخفيف آثار أو تجنب حدوث تلك المخاطر. ومثال ذلك أنه يحتمل حدوث مخاطر في أثناء مهمة تكامل الأنظمة الفرعية للمشروع نتيجة وجود شخص واحد فقط عنده المقدرة على إنجاز تلك المهمة، وهنا يمكن لمدير المشروع اتخاذ أي من الأفعال التالية لتخفيف حدة أو تجنب هذه المخاطرة:

- توظيف شخص إضافي يمتلك الخبرة المطلوبة واعتباره بديلاً ومساعداً للشخص الأساسي.
 - عرض مزايا إضافية للشخص الوحيد المتوافر حالياً لضمان بقائه.
 - استخدام طريقة أخرى لإنجاز المهمة بحيث لا تحتاج لتلك المهارة الخاصة.
- ويعنى هذا أنه توجد غالباً عدة بدائل لتخفيف المخاطر، ولكن اتخاذ القرار باختيار أحدها يعتمد على عدة عوامل منها:
- سهولة أسلوب التخفيف.
 - احتمالية نجاح الأسلوب.
 - تكلفة الأسلوب.

فيمكن لمدير المشروع بعد تحديد بدائل الحلول لتخفيف حدة المخاطر والتغلب عليها، أن يقدر تكلفة كل بديل وبعد ذلك يتم اختيار البديل الأقل تكلفة. ولكن مع الأخذ في الاعتبار احتمالية نجاح كل بديل، يمكن أن يؤدي ذلك إلى اختلاف القرار. فمن الممكن أن يكون البديل الأقل تكلفة هو أيضاً الأقل احتمالية لنجاح تطبيقه، ويكون بديل آخر أكثر تكلفة منه ولكن درجة احتمالية نجاحه أكبر، وهنا يكون القرار الأفضل هو اختيار البديل الأكثر تكلفة والأكثر احتمالية للنجاح. ويمكن استخدام الصيغة التالية التي تجمع بين العاملين لتحديد قيمة تكلفة تخفيف المخاطر:

$$\text{تكلفة تخفيف المخاطرة} = \text{تكلفة الأسلوب} * \text{احتمالية فشل الأسلوب}$$

خطة إزالة المخاطر Risk Removal Plan

يقوم مدير المشروع بعد ذلك بإعداد خطة لإزالة المخاطر بناء على المخاطر المحددة والمرتبة والأفعال والتدابير التي تم اختيارها لتجنب حدوثها. يوضح شكل (٤-١٢) مثالاً لخطة إزالة المخاطر. تحتوى الخطة على قائمة المخاطر مرتبة حسب أولوية حدوثها، وكذلك التاريخ المتوقع لإزالتها، مع ذكر الأفعال التي سيتم اتخاذها لإزالة المخاطر.

الأفعال المطلوبة لإزالة المخاطر	التاريخ المتوقع لإزالة المخاطر	قائمة المخاطر (مرتبة)
إنجاز عمليات تخفيف المخاطرة	٢٠٠٦/٠٦/٣٠	مخاطرة ١
إنجاز المهمة رقم ٣	٢٠٠٦/٠٧/١٦	مخاطرة ٢
إنجاز المهمة رقم ٨، وإتمام عمليات تخفيف المخاطرة	٢٠٠٦/٠٩/٢١	مخاطرة ٣
.	.	.
.	.	.
.	.	.

شكل (٤-١٢) خطة إزالة المخاطر

يجب على مدير المشروع مراجعة خطة إزالة المخاطر باستمرار مع تقدم الإنجاز في المشروع، وذلك بحذف مخاطر من القائمة أو إضافة مخاطر أخرى طرأت حديثاً أو تغيير خصائص بعض المخاطر.

٧-٤ إدارة الجودة Quality Management

يعتبر نشاط إدارة الجودة من الأنشطة التي تتضافر مع أنشطة إدارة المشروع وكذلك عمليات التطوير المختلفة للمشروع للتأكد من جودة البرمجيات المنتجة ومطابقتها لمواصفات الجودة المطلوبة. ولذلك فإن إدارة الجودة تكون غالباً مستقلة عن إدارة المشروع ولها فريق خاص غير فريق التطوير ولها ميزانية وخطة عمل مستقلة. هناك معايير قياسية دولية تستخدم لتقييم جودة البرمجيات للمؤسسات المنتجة لها وهي معروفة باسم "ISO 9000 Series".

تتضمن عملية إدارة الجودة الأنشطة التالية:

- إعداد خطة الجودة Quality Plan.
- مراقبة الجودة Quality Control.
- ضمان الجودة Quality Assurance.

٤-٧-١ جودة البرمجيات Software Quality

تعتمد الجودة المطلوبة على طبيعة مشروع البرمجيات، التي تركز على نقاط جودة أكثر من غيرها. يتم تصنيف جودة البرمجيات إلى نوعين أساسيين هما:

- جودة المنتج Product Quality.

- جودة العملية Process Quality.

مع الأخذ في الاعتبار أنه لا يمكن الحصول على منتج جيد بدون التأكد من جودة العمليات التي تمت لإنتاجه. ولهذا فإن إدارة الجودة تتطلب التأكد من جودة المنتج البرمجي خلال مراحل تطويره وليس بعد استلامه فحسب. تحتاج هذه العملية لاستخدام أداة إدارة التشكيل Configuration Management Tool لدعم عمليات تخزين ومعالجة بيانات النسخ المطورة للمنتج البرمجي خلال مراحل تطويره. هناك العديد من الخصائص والمعايير التي تصف جودة المنتج البرمجي وهي (Ghezzi, 2003):

- الصحة Correctness.

- الثقة Reliability.

- المتانة Robustness.

- الأداء Performance.

- الاستخدام Usability.

- الفهم Understandability.

- قابلية الصيانة Maintainability.

- قابلية التطور Scalability.

- إعادة الاستخدام Reusability.

- قابلية النقل Portability.

- إمكانية التشغيل البيني Interoperability.

- الإنتاجية Productivity.

- الرؤية Visibility.

٤-٧-٢ مراقبة الجودة Quality Control

تركز عملية مراقبة الجودة على وجود اختلافات بين مواصفات العمليات أو المنتجات البرمجية بالمقارنة بمواصفات المتطلبات. ولذلك فإن الهدف من عملية مراقبة الجودة هو إجراء العديد من الاختبارات على العمليات التي تؤدي إلى إنتاج المنتجات البرمجية للحد من المشكلات التي تهدد جودة المنتج في وقت مبكر. فقد أكدت دراسة سابقة لشركة IBM أن تكلفة تصحيح الأخطاء الناجمة عن عمليات الاختبار ومراقبة الجودة تختلف بشدة حسب المرحلة التي تم اكتشاف الأخطاء فيها. فهي تكون قليلة عند اكتشافها مبكراً وتتراكم وتزداد بشكل غير خطي عند اكتشافها في مراحل متأخرة. وعلى هذا فإن عملية الاختبار يجب أن تتم خلال مراحل التطوير المختلفة ويتم جدولتها بشكل منفصل ويقوم بها فريق مؤهل وتكون لها ميزانية منفصلة. تستخدم عملية مراقبة الجودة العديد من أساليب الاختبار التي تتضمن معظم المعايير التي تم سردها سابقاً لوصف جودة المنتج البرمجي. من أهم هذه الاختبارات ما يلي:

١- اختبار الصندوق الأسود Black Box Test

نوع من الاختبارات يقوم على مبدأ عدم الاهتمام بمحتوى الوحدة البرمجية تحت الاختبار والتعامل معها باعتبارها صندوقاً أسود. ولذلك فإن هذا النوع من الاختبارات يناسب مجموعة المستخدمين للنظام. تبدأ عملية الاختبار بتغذية الوحدة البرمجية ببيانات الاختبار الموجودة في حالات الاختبار المعدة مسبقاً، وبعد ذلك يتم مقارنة النتائج بما هو متوقع في حالات الاختبار. يهتم هذا النوع من الاختبارات باعتباريات صحة وجودة المتطلبات الوظيفية للمنتج البرمجي وهو يعتبر مؤشراً لقبول أو رفض المنتج من ناحية مستخدم النظام. يستخدم اختبار الصندوق الأسود لاكتشاف الأخطاء التالية:

- وظائف غير صحيحة أو ناقصة.
- أخطاء في واجهات استخدام النظام.
- أخطاء في هياكل البيانات أو قواعد البيانات.
- أخطاء في الأداء.

٢- اختبار الصندوق الأبيض White Box Test:

على النقيض من اختبار الصندوق الأسود، فإن اختبار الصندوق الأبيض يركز على اختبار محتوى المنتج البرمجي. ويقوم بتنفيذه مجموعة من المتخصصين في اختبار البرمجيات. يبدأ المختبر عادة بمراجعة نماذج التصميم ومواصفات المتطلبات وبعد ذلك يبدأ بدراسة الأكواد وتتبع الأخطاء الموجودة وتصحيحها. ولذلك فإن طبيعة اختبار الصندوق الأبيض تعتمد على المراجعة والتدقيق والفحص المستمر لمراحل تطوير المنتج البرمجي بهدف اكتشاف الأخطاء وتصحيحها في مرحلة مبكرة. جدير بالذكر أن هذا النوع من الاختبارات يعتمد بشكل أساسي على خبرة أفراد فريق الاختبار في استخدام أساليب الاختبار بشكل جيد.

٣- الاختبار الآلي Automated Test:

نوع من الاختبارات يتم إجراؤه بشكل آلي باستخدام أدوات اختبار البرمجيات، وهو أسلوب يناسب التطبيقات الكبيرة الحجم التي يتم إعدادها بشكل قياسي باستخدام أدوات هندسة البرمجيات المدعومة بالحاسب CASE Tools.

خطة الاختبار Test Plan:

تعتبر خطة الاختبار جزءاً أساسياً من خطة إدارة الجودة وهي تحدد أنشطة جدولية عملية الاختبار وإعداد حالات الاختبار وتحديد الميزانية المطلوبة لتنفيذ الاختبار. تتضمن عملية التخطيط للاختبار الأنشطة التالية:

- تحديد المهام التي سيتم اختبارها.
 - تحديد الأفراد الذين يقومون بتنفيذ الاختبار.
 - توزيع مهام الاختبار على الأفراد.
 - إعداد حالات الاختبار من واقع مواصفات متطلبات النظام.
- تحتوي كل حالة اختبار على مواصفات الأجهزة والبرمجيات التي تناسب عملية الاختبار وكذلك المحاولات المقترحة لتنفيذ الاختبار. يتم تحقيق هذه المواصفات في شكل سيناريو مكتوب يحتوي على جميع مواصفات حالات الاختبار وخطوات إجراء الاختبار وكذلك نقاط التحقق Verification Points.

٤-٧-٣ تأمين الجودة Quality Assurance،

تهدف عملية تأمين جودة البرمجيات إلى التحقق من مطابقة المنتجات البرمجية لمعايير الجودة التي تم إقرارها في أهداف المشروع. يتم ذلك من خلال مراقبة خطة المشروع كاملة مثل الجدولة والميزانية وتخصيص الموارد واستخدامها، بالإضافة إلى مراقبة إجراءات تطوير المنتج البرمجي. يتم استخدام مجموعة من الأساليب التي تحقق تأمين جودة البرمجيات مثل:

١- قوائم التدقيق Checklists،

تحتوي قائمة التدقيق على مجموعة من الأسئلة والمواصفات التي تستخدم في عمليات التحقق من جودة العملية أو المنتج البرمجي، وهي بذلك تمثل معايير الجودة المبدئية للمنتج البرمجي. يمكن تجهيز قائمة التدقيق من خلال مجموعة من نقاط التدقيق المرتبة حسب الأولوية وأهميتها للعملية البرمجية. تخضع قائمة التدقيق للمعايير القياسية الخاصة بهندسة البرمجيات وهي تضمن مطابقتها لمواصفات الجودة العالمية ISO.

٢- مراجعات Reviews،

تتم مراجعة العمليات والمنتجات البرمجية من خلال اجتماعات عمل تضم المطورين للعمليات التي يتم مراجعتها ومراجعين وقائد فريق المراجعة، وذلك للتأكد من جودة العمليات المنتجة. يتم تزويد أعضاء فريق المراجعة بالعمليات المنجزة لمراجعتها وتدوين الملاحظات عليها قبل الاجتماع. يبدأ مطور العملية التي يتم مراجعتها بعرض سريع للإجراءات المنجزة مع السماح لأعضاء المراجعة بالتعليق عليها. يتم تسجيل جميع الملاحظات في وثيقة المراجعة التي يتم تسليمها لمطور العملية في نهاية الاجتماع ليقوم بإجراء التصحيحات المطلوبة. بعد انتهاء عملية التصحيح يتم إبلاغ قائد فريق المراجعة بذلك ليقوم بمراجعتها واتخاذ القرار بقبول العملية أو الإعداد لاجتماع مراجعة آخر.

٣- تدقيقات Audits،

يعتبر أسلوب التدقيق أكثر شمولاً من أسلوب القوائم وأسلوب المراجعة، إذ يبدأ بعملية تجهيزية تقوم بدراسة المنتج المطلوب تدقيقه، وبعد ذلك يتم إعداد مجموعة

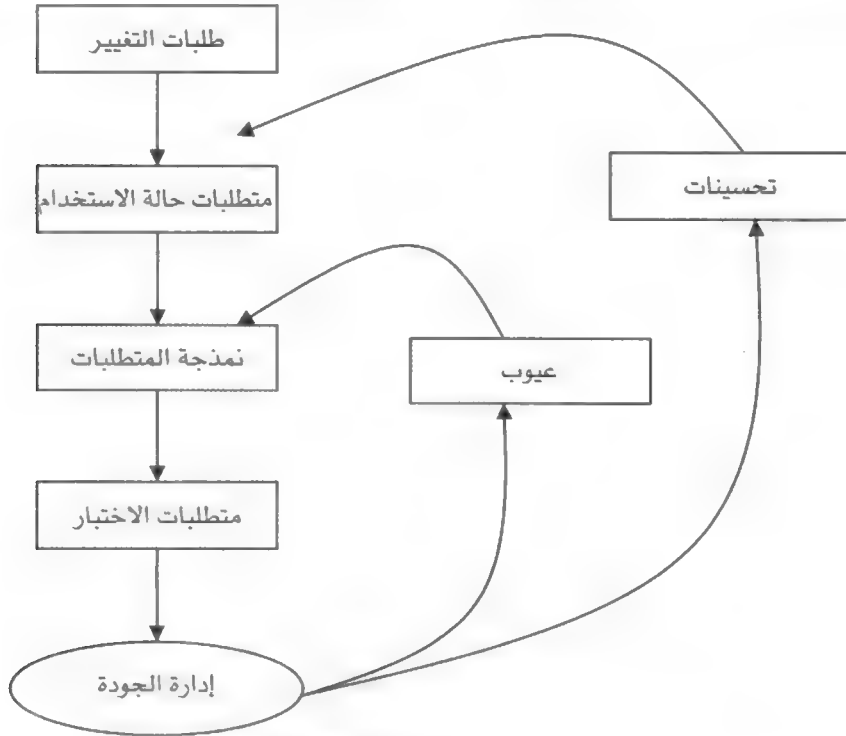
من الاستفسارات فى شكل قائمة تدقيق Checklist يتم فحص العملية أو المنتج من خلالها. يتبع ذلك إجراء مقابلات مع فريق التدقيق، إذ يتم تجميع معلومات التدقيق التى تم تدوينها فى أثناء الفحص ومراجعتها، وبعد ذلك يتم توثيق خلاصة عملية التدقيق فى شكل تقرير فحص للمنتج البرمجى يحتوى على جميع الملاحظات الفنية على المنتج.

٨-٤ إدارة التشكيل والتغيير Change and Configuration Management

تهتم عملية إدارة التشكيل والتغيير بتنظيم عملية تطوير المنتج البرمجى وإدارة فريق العمل القائم على عملية التطوير. تشمل هذه العملية مراحل تطوير المنتج البرمجى وما يتعرض له من تغييرات. يوضح الشكل (٤-١٣) المفهوم العام لعملية إدارة التشكيل والتغيير. يؤدى طلب التغيير غالباً إلى تعديل فى مواصفات حالات الاستخدام وهو يؤدى بدوره إلى تعديل نماذج المتطلبات، وبعد ذلك يتم اختبار صلاحية المتطلبات بعد إجراء التعديلات والتحقق من جودة نماذج المتطلبات وتصحيحها أو إجراء تحسينات عليها وإعادة اختبارها وتقييمها.

١-٨-٤ التغيير فى المتطلبات Requirements Change

تتسم معظم مشاريع البرمجيات بإمكانية طلب المستخدم إجراء تغييرات فى متطلبات النظام فى أثناء عملية التطوير. أحياناً تكون بعض التغييرات جوهرية وذات تأثير مباشر فى خطة المشروع، وأحياناً تكون التغييرات يسيرة يمكن إدراجها فى متطلبات النظام من غير تأثير واضح فى خطة المشروع. ولهذا فإنه يجب التعامل مع طلبات التغيير بشكل رسمى من خلال عملية مخصصة لدراسة التغيير المطلوب ومدى تأثيره فى خطة المشروع الحالية ومن ثم قبوله أو رفضه. تبدأ هذه العملية بصياغة مواصفات التغيير المطلوب فى شكل طلب تغيير متطلبات. يحتوى هذا الطلب على بعض التوصيات حول التغيير المقترح بالإضافة إلى البيانات الخاصة بمواصفات التغيير وتأثيره فى خطة المشروع من حيث الوقت والميزانية. يتم دراسة الطلب من قبل مجموعة من الفنيين وتقييم التغيير المقترح من ناحية تأثيراته المتوقعة فى المشروع، ومن ثم تقرير مدى إمكانية تحقيق التغيير أو التوصية برفض طلب التغيير. يتم رفع هذا التقرير إلى مدير المشروع للموافقة على التوصيات أو إعادة الدراسة مرة أخرى. وفى حالة الموافقة عليه يتم إصدار تقرير بمواصفات التغيير المطلوب يتضمن مجموعة القيود ومعايير الجودة التى تحكم عملية تطبيق التغيير.



شكل (١٣-٤) مفهوم عملية إدارة التشكيل والتغيير

٢-٨-٤ إصدارات المنتج Product Versions

تؤدي عمليات التطوير خلال المراحل المختلفة إلى إنتاج مستلزمات معينة سواء في شكل نماذج أو برامج أو واجهات استخدام. ويؤدي أسلوب التطوير بالتكرار والزيادة لإنتاج أكثر من نسخة للمستلم أو المنتج. يتم تخزين جميع نسخ منتجات النظام في مستودع رئيسي Repository ويتم التعامل معها من خلال إدارة نظام إدارة التشكيل Configuration Management system. يمكن التعديل على هذه النسخ في أثناء تنفيذ طلبات التغيير في المتطلبات ليتم الحصول على نسخ أخرى مطورة للمنتج، وهكذا حتى يتم الحصول على الإصدار الأخير للمنتج البرمجي. تكمن الاستفادة من المستودع في إمكانية استخدام إحدى النسخ البديلة للمنتج في مرحلة سابقة واستكمال عمليات التطوير عليها لإصدار نسخة مطورة للمنتج البرمجي. وبعد انتهاء عمليات الاختبار للنسخة المطورة للمنتج فإنه يتم الانتقال إلى مرحلة إصدار المنتج،

إذ يأخذ رقم إصدار ويتم تثبيته عند العميل لبدء التشغيل الفعلي. تتضمن عمليات نشر وتثبيت الإصدار للمنتج البرمجي المستلمات التالية:

- ملفات تنفيذية Executable Files.
- برامج قواعد البيانات على الخادم Server Database Programs.
- ملفات قواعد البيانات Database Files.
- ملفات التشكيل Configuration Files.
- ملفات التوثيق System Documentations.
- ملفات المساعدة Help Files.
- تعليمات التثبيت Installation Instructions.

٤-٨-٣ العيوب والتحسينات Defects and Enhancements:

تظهر عيوب المنتجات البرمجية في عدة مراحل في أثناء عمليات المراجعة أو في أثناء الاختبار أو في أثناء التشغيل الفعلي عند العميل. وتكون التحسينات على المنتج في شكل توصيات مقترحة بإضافة سمات جديدة أو إجراء تعديلات في الإصدار الحالي لإنتاج إصدار جديد محسن يفي بالمتطلبات الجديدة ولا يحتوى على العيوب الموجودة بالنسخة الحالية للمنتج. تكون هذه التوصيات المقترحة نتيجة لطلب مباشر من أحد المشاركين في النظام بإجراء تعديلات أو إضافة متطلبات للنسخة الحالية وذلك بدون حدوث أى عيوب أو مشكلات في الإصدار الحالي للمنتج. ويمكن أيضاً أن تكون التوصيات المقترحة نتيجة لحدوث عيوب ومشكلات في الإصدار الحالي ويجب علاجها وإصدار نسخة محسنة للإصدار الحالي. تأخذ العيوب التي يمكن ظهورها في المنتجات البرمجية عدة أشكال أو حالات في أثناء عملية معالجتها مثل:

- عيوب جديدة New.
- عيوب تم حلها Resolved.
- عيوب غير مؤكدة Unconfirmed.
- عيوب تم ظهورها مجدداً Reopened.

- عيوب تم تخصيصها Assigned.
- عيوب معلقة Suspended.
- عيوب فى حالة انتظار Waiting.
- عيوب تم إغلاقها Closed.

يقوم فريق المراجعة والاختبار بتحديد نوعية العيوب الموجودة وتقييم درجة خطورتها على المنتج ويمكن تقييم درجة خطورة العيوب المكتشفة فى المستويات التالية:

- بديهية Trivial.
- بسيطة Minor.
- عادية Normal.
- رئيسية Major.
- حرجة Critical.
- عائقة Blocker.

وبعد تحليل العيوب الناجمة، يتم اتخاذ الأفعال المناسبة للتغلب عليها فى شكل مجموعة من الإجراءات، ويأخذ تقرير الأفعال المطلوبة أحد الأشكال التالية:

- غير صحيحة Invalid.
- مكررة Duplicate.
- جارى العمل فيها Workaround.
- تم حلها Fixed.
- لا يمكن حلها Wont fix.

الجزء الثانى

تطوير مشاريع البرمجيات

Software Projects Development

نقدم فى هذا الجزء عرضاً للعمليات التى يتم تنفيذها فى أثناء تطوير مشاريع البرمجيات. وسوف نعتد أحد الأساليب الحديثة لتطوير البرمجيات وهو أسلوب التكرار مع الزيادة Iterative with Increment الذى تم عرضه فى الجزء الأول من هذا الكتاب. يتميز هذا الأسلوب بمساعدة كل من فريق التطوير ومستخدمى النظام للمشاركة الفعالة خلال مراحل التطوير التى تضمن إنتاج برمجيات تحقق متطلبات المستخدمين. يتم ذلك من خلال إنتاج البرمجيات المطلوبة بالتدرج وذلك بعد تحديد أولويات متطلبات النظام من قبل المستخدمين. فتبدأ عمليات التطوير دائماً بتحليل المتطلبات الضرورية للنظام، ثم تصميم الحلول لها، ثم تطبيقها واختبارها، وبعد ذلك يتم تسليمها للمستخدم فى شكل نسخة أولية للنظام. يقوم فريق المستخدمين بتجربة النسخة الأولية للنظام وإبداء الملاحظات ومناقشتها مع فريق التطوير لإجراء التعديلات المطلوبة مبكراً. وبعد ذلك يتم إضافة متطلبات أخرى للنسخة المطورة الحالية وإعادة عمليات التطوير حتى يتم الحصول على نسخة أخرى موسعة تحتوى على وظائف إضافية للنسخة الحالية. يتم تكرار هذا الأسلوب حتى يتم إنتاج النسخة النهائية للنظام التى تتضمن جميع متطلبات النظام وذلك بمشاركة معظم المساهمين فى مشروع البرمجيات. يؤدى هذا الأسلوب إلى إنتاج برمجيات ذات جودة عالية تحوز ثقة العميل والمستخدم.

يشمل هذا الجزء عرضاً وافياً لعمليات تطوير النظم مثل:

- تحليل متطلبات النظام System Requirements Analysis.
- التصميم المعمارى للنظام Architectural System Design.
- تصميم قواعد البيانات Database Design.
- برمجة واختبار التطبيق Programming and Testing.
- تصميم واجهات المستخدم User Interface Design.

وسوف نستعين ببعض التطبيقات والحالات الدراسية لدعم عملية عرض أنشطة تطوير مشاريع البرمجيات.

الفصل الخامس

تحديد المتطلبات

Requirements Determination

يتضمن أى مشروع برمجيات مجموعة من المتطلبات والاحتياجات التى تصف ما يجب أن يقوم به النظام وظروف التشغيل. يتم تصنيف المتطلبات إلى نوعين هما:

١- متطلبات وظيفية؛

تصف مجموعة الخدمات والوظائف التى يجب أن يوفرها النظام.

٢- متطلبات غير وظيفية؛

تصف مجموعة القيود التى تحكم تشغيل النظام.

تحتوى عملية تحديد متطلبات مشروع البرمجيات على العديد من الأنشطة مثل:

- تجميع المعلومات.
 - تعريف المتطلبات الوظيفية وغير الوظيفية.
 - تحديد أولويات المتطلبات.
 - تحليل المتطلبات.
 - توثيق المتطلبات.
- وسوف نركز فى هذا الفصل على النشاط الخاص بعمليات تحليل المتطلبات.

١-٥ نموذج كائنات نظام الأعمال Business Object Model؛

يعتبر نموذج كائنات نظام الأعمال BOM بمنزلة إطار العمل Framework الخاص بعمليات تطوير نظام الأعمال بالمؤسسة. وهو يستخدم أيضاً وسيلة اتصال فعالة بين فريق التطوير والمستخدمين فى أثناء عمليات التطوير. وكما ذكرنا سابقاً فإن استخدام أسلوب النمذجة المرئية يساعد كثيراً فى توصيل المفهوم فى أثناء عملية

تحليل المتطلبات. وهو أسلوب يمكننا من وصف النظام من خلال عدة منظورات ويصف جميع العلاقات والتفاعلات الموجودة داخل نظام الأعمال. يصف هذا النموذج مجموعة الكائنات الأساسية لنظام الأعمال والعلاقات بينها في وضع السكون أو ما يسمى بالمنظور التركيبي Structural View لنظام الأعمال. ويصف كذلك تفاعل نظام الأعمال مع البيئة المحيطة به من خلال وصف مجموعة الفاعلين Actors وتفاعلهم مع نظام الأعمال من خلال مجموعة من حالات الاستخدام Use Cases. ويمكننا أيضاً استخدام أسلوب التحليل الهيكلي للمتطلبات عن طريق استخدام مخطط السياق Context Diagram ومخططات تدفق البيانات Data Flow Diagrams (DFDs).

١-١-٥ نظام الأعمال The Business

تعتمد صحة وجودة عملية تحديد وتحليل المتطلبات على مدى فهم فريق التطوير لنظام الأعمال. تشمل عملية الفهم عدة جوانب مهمة مثل:

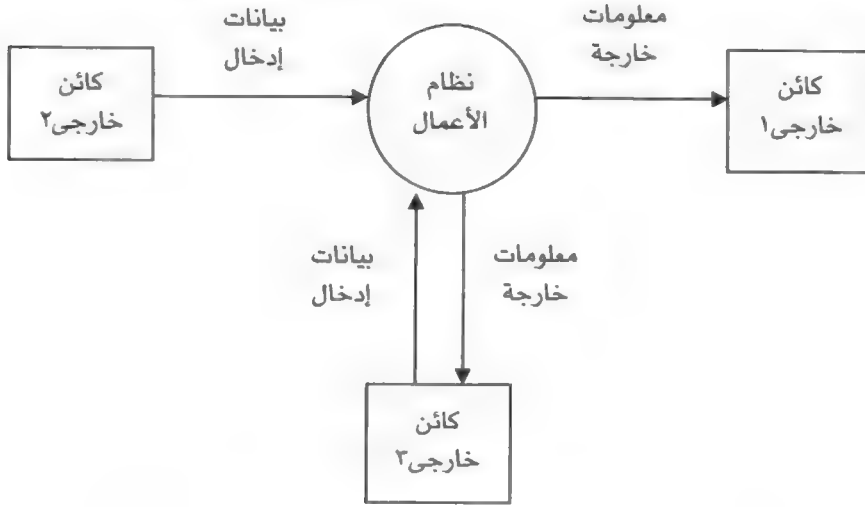
- أهداف نظام الأعمال Business Objectives.
- مجالات نظام الأعمال Business Domains.
- الهيكل التنظيمي لنظام الأعمال Organizational Structure.
- الأنظمة الفرعية داخل نظام الأعمال والتفاعلات بينها Interrelated Subsystems.
- مجموعة الفاعلين لنظام الأعمال Actors.
- المتطلبات الوظيفية لنظام الأعمال Functional Requirements.
- المتطلبات غير الوظيفية لنظام الأعمال Nonfunctional Requirements.

تتم عملية فهم نظام الأعمال وتحديد متطلباته من خلال عقد لقاءات عمل مع كل من العميل ومستخدمي النظام. يقوم العميل أو صاحب نظام الأعمال أو المسؤول وصاحب القرار بتحديد الأهداف العامة لنظام الأعمال وتحديد مجموعة متطلبات النظام غير الوظيفية مثل: الجودة والكفاءة المطلوبة وزمن التنفيذ والميزانية المخصصة وطريقة التسليم ومعايير القبول ومتطلبات السلامة ومتطلبات أخرى قانونية وغيرها من المتطلبات غير الوظيفية. أما مواصفات متطلبات نظام الأعمال الوظيفية فيتم

معرفتها وفهمها من خلال لقاءات دورية مجدولة مع مستخدمي النظام في المستويات المختلفة بدءاً من مديري العموم مروراً برؤساء الأقسام ووصولاً إلى المستخدم النهائي لمهام نظام الأعمال. يقوم كل مستخدم حسب موقعه ومسؤوليته بشرح ووصف طبيعة المهام التي يقوم بها ووصف للبيانات التي يستقبلها ومصدرها والمعلومات التي ينتجها والجهات التي تصدر إليها. بالإضافة إلى عرض للمشكلات الحالية بالنظام والمقترحات لحلها إذا أمكن وكذلك التحسينات المطلوبة من المستخدم على إجراءات تنفيذ المهام. يتم الاستعانة بمجموعة الوثائق والمستندات واللوائح والتقارير التي يستخدمها ويتعامل معها مستخدمو النظام في أثناء إنجاز أعمالهم. ومن الممكن الاستعانة بأفراد أو جهات خارجية ذات خبرة في مجال نظام الأعمال للتأكد من صحة ودقة ومصادقية المعلومات والمتطلبات المجمعّة ومدى إمكانية تحقيقها في الوقت المطلوب وبالميزانية المخصصة.

٥-١-٢ مخطط السياق لنظام الأعمال Business Context Diagram

يصف هذا المخطط حدود نظام الأعمال العام داخل البيئة المحيطة به. يتم تمثيل ذلك من خلال نشاط عام يمثل نظام الأعمال العام وتفاعلاته المختلفة مع الكائنات الخارجية التي تتعامل مع نظام الأعمال. تمثل هذه الكائنات الخارجية مجموعة الأفراد أو المؤسسات أو الأنظمة الأخرى التي تتعامل مع نظام الأعمال. يتم تمثيل التفاعلات بين الكائنات الخارجية ونظام الأعمال من خلال مجموعة من تدفقات البيانات Data Flows. تمثل هذه التدفقات البيانات التي يستقبلها نظام الأعمال من الكائنات الخارجية وكذلك المعلومات التي يرسلها نظام الأعمال إلى الكائنات الخارجية. وعلى هذا الأساس يتم تمثيل نظام الأعمال في مخطط السياق على شكل عملية أو نشاط عام يمثل مجموعة العمليات والإجراءات التي تتم على البيانات المدخلة للنظام لإنتاج المعلومات التي سيتم إمدادها إلى الكائنات المحيطة بالنظام للاستفادة منها. يوضح شكل (٥-١) الشكل العام لمخطط السياق لنظام الأعمال. نلاحظ أن بعض الكائنات الخارجية تكون مرسلة للبيانات فقط أو مستقبلة لها والبعض الآخر يكون مرسلًا ومستقبلًا للبيانات.



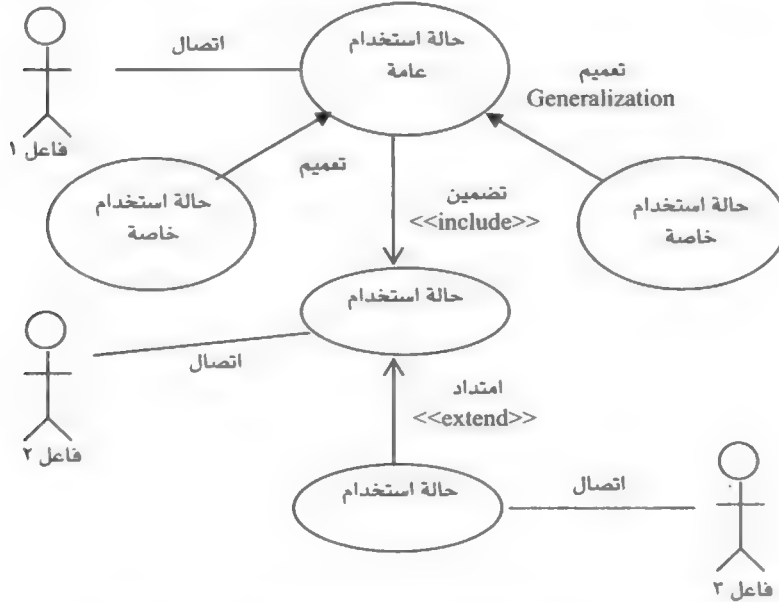
شكل (١-٥) مخطط السياق لنظام الأعمال

ومع أن مخطط السياق ليس من المخططات المعتمدة في لغة UML، ولكن من الأفضل أن يتم البدء به في عملية نمذجة المتطلبات، إذ إنه يعطى صورة واضحة ومفيدة لكيفية تعامل النظام مع البيئة المحيطة به من خلال مجموعة التفاعلات مع مستخدمي النظام.

٣-١-٥ مخطط حالات الاستخدام لنظام الأعمال Business Use Case Diagram

يمثل مخطط حالات الاستخدام لنظام الأعمال المستوى الأول لتجزئة مخطط السياق لنظام الأعمال. يمكننا بعد ذلك تجزئة هذا المخطط العام إلى مجموعة من مخططات حالات الاستخدام الفرعية للنظام. وعلى هذا فهو يمثل حالات استخدام نظام الأعمال في المستوى الأعلى في شكل وحدات وظيفية رئيسية ذات حدود واضحة تمثل النشاطات الأساسية لنظام الأعمال. يحتوى نموذج حالات الاستخدام للنظام على وصف لمجموعة الفاعلين Actors الذين يتفاعلون مع حالات الاستخدام. يمثل الفاعل مجموعة الأشخاص أو الأنظمة التي تتعامل مع الخدمات التي تمثلها حالات الاستخدام. يمكن أن يمثل الفاعل الكائنات الخارجية لنظام الأعمال أو يمثل دوراً داخلياً مباشراً داخل النظام مثل مستخدمى النظام. ويحتوى نموذج حالات الاستخدام للنظام أيضاً على مجموعة من حالات استخدام النظام التي يمكن أن ترتبط معا

بـعـلـاقـات اـرتـبـاط Association أو عـلـاقـات تـعـمـيم Generalization. يـوضـح شـكـل (٢-٥) نـمـوـذـجـاً عـامـاً لـمـخـطـط حـالـات اسـتـخـدام لـنـظـام الأـعـمال.



شكل (٢-٥) نموذج لمخطط حالات الاستخدام لنظام الأعمال

٤-١-٥ مخطط الأصناف لنظام الأعمال Business Class Diagram

يوضح هذا النموذج مجموعة الكائنات الأساسية لنظام الأعمال والعلاقات بينها. يسهم كل كائن أساسي في تقديم خدمة أو أكثر من الخدمات الأساسية التي يقدمها نظام الأعمال. يتم تمثيل أصناف نظام الأعمال بشكل مختلف عن أصناف المجال، فيتم تمثيلها من خلال دائرة تحتها خط أفقي وبداخلها خط مائل. ومع أن الصنف يحتوي على مجموعة من الخصائص والعمليات ولكن لا يتم تمثيلها في نموذج أصناف نظام الأعمال؛ لأنه يمثل الكائنات الأساسية في المستويات العليا لنظام الأعمال وبشكل مجرد. هناك ثلاثة أنواع من العلاقات يمكن تمثيلها بين الأصناف العليا هي:

- الارتباط Association.
- التعميم Generalization.
- التجميع Aggregation.

يمكننا أيضاً توضيح درجة التعددية Multiplicity لكل طرف في العلاقة بين الأصناف مثل علاقات:

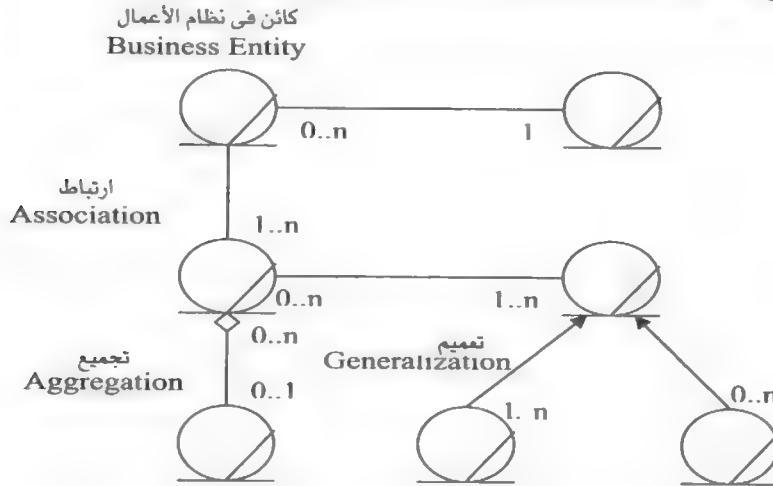
واحد- إلى - واحد One-to-One.

واحد- إلى - متعدد One-to-Many.

متعدد- إلى - متعدد Many-to-Many.

ويمكن أن تكون العلاقة اختيارية Optional عندما يكون من الممكن أن يشارك الكائن في العلاقة بإحدى حالاته أو لا يشارك بأى حالة. ويمكن أن تكون العلاقة إجبارية Mandatory بأن يشارك الكائن في العلاقة بأكثر من حالة أو يشارك على الأقل بحالة واحدة.

يوضح شكل (٣-٥) نموذجاً لمخطط الأصناف لنظام الأعمال العام.



شكل (٣-٥) نموذج لمخطط الأصناف لنظام الأعمال العام

٢-٥ نموذج كائنات المجال Domain Object Model:

يعتبر نموذج كائنات المجال DOM هو أحد النماذج الفرعية لنموذج كائنات نظام الأعمال BOM. أى أنه يوجد دائماً نموذج واحد يصف كائنات النظام BOM و عدة نماذج تصف كائنات المجال DOMs. ونجد أيضاً أن كل حالة استخدام عامة في

نموذج كائنات النظام يتم تمثيلها من خلال نموذج كائنات لأحد مجالات النظام. والمجال هو أحد التطبيقات أو النظم الفرعية للنظام العام لمشروع الأعمال. والمجال له حدود واضحة متمثلة في شكل مجموعة محددة من حالات الاستخدام والفاعلين المشاركين فيها. وعلى هذا فإن عملية النمذجة سيتم تطبيقها مباشرة على مستوى كائنات المجال.

٥-٢-١ المجال Domain:

كما ذكرنا سابقاً فإن أى نظام أعمال يحتوى على مجموعة من الوحدات الوظيفية الفرعية التى تتضمن مجموعة الخدمات والوظائف التى يقدمها نظام الأعمال. ومن الطبيعى أن تتبع كل وحدة فرعية أحد المجالات التى يتضمنها نظام الأعمال العام. فمثلاً يعتبر نظام الإجازات أحد المجالات الفرعية للنظام العام لشؤون الموظفين بأى مؤسسة. وهو يحتوى على مجموعة الوظائف والعمليات وقواعد العمل التى تحكم عملية منح الإجازات للموظفين وكذلك تفاعلها مع مجالات أخرى داخل النظام العام مثل تأثير الإجازات فى الرواتب والترقيات والابتعاث وغيرها من النظم الفرعية داخل نظام شؤون الموظفين.

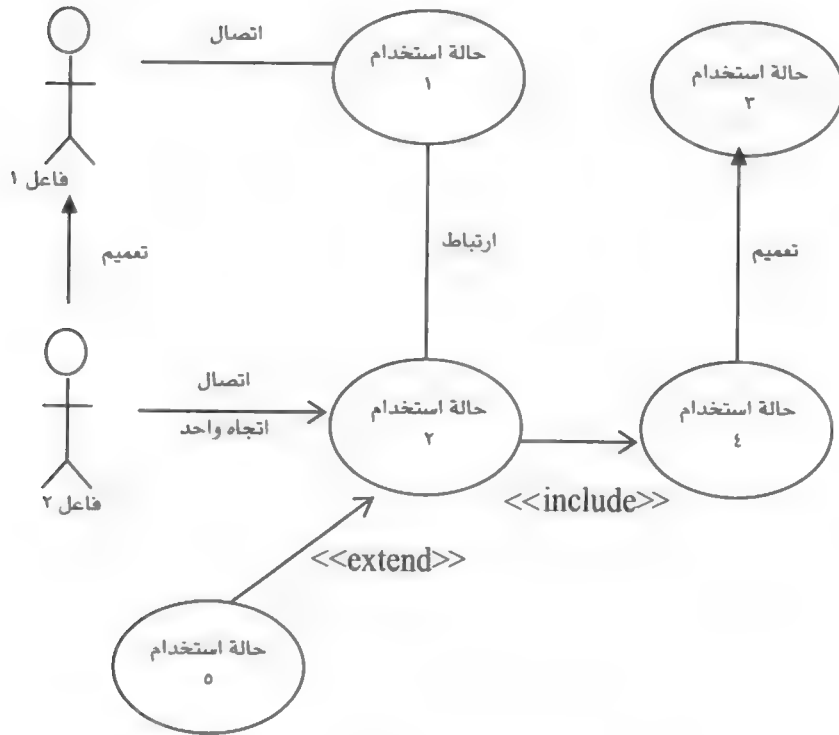
٥-٢-٢ مخطط حالات الاستخدام للمجال Domain Use Case Diagram:

تعتبر مخططات حالات الاستخدام لمجالات النظام بمنزلة الوصف التفصيلى لمخطط حالات الاستخدام لنظام الأعمال العام. يمثل كل مخطط تفصيلى مجموعة الوظائف لأحد مجالات النظام والعلاقات بينها وبين البيئة المحيطة به. تمثل كل حالة استخدام Use Case وظيفة رئيسية من وظائف أحد مجالات النظام وتتفاعل مع الفاعلين Actors الذين يصفون بيئة المجال. يمكن أن يحتوى المجال على عدة مخططات لحالات الاستخدام، وأحياناً يتم وصف جميع حالات المجال فى مخطط واحد. ترتبط حالات الاستخدام معاً فى شكل علاقات ارتباط Association أو علاقات تعميم Generalization أو علاقات تضمين Inclusion أو علاقات امتداد Extension. ويرتبط الفاعل Actor بحالات الاستخدام من خلال علاقة اتصال Communication. جدير بالذكر أنه لا يمكن الاعتماد على مخطط حالات الاستخدام لوصف وظائف المجال فحسب، إنما يتم الاستعانة بملفات نصية Text Documents ترفق مع المخطط لتوثيق مواصفات حالات الاستخدام التى يتم تخزينها مع مخطط

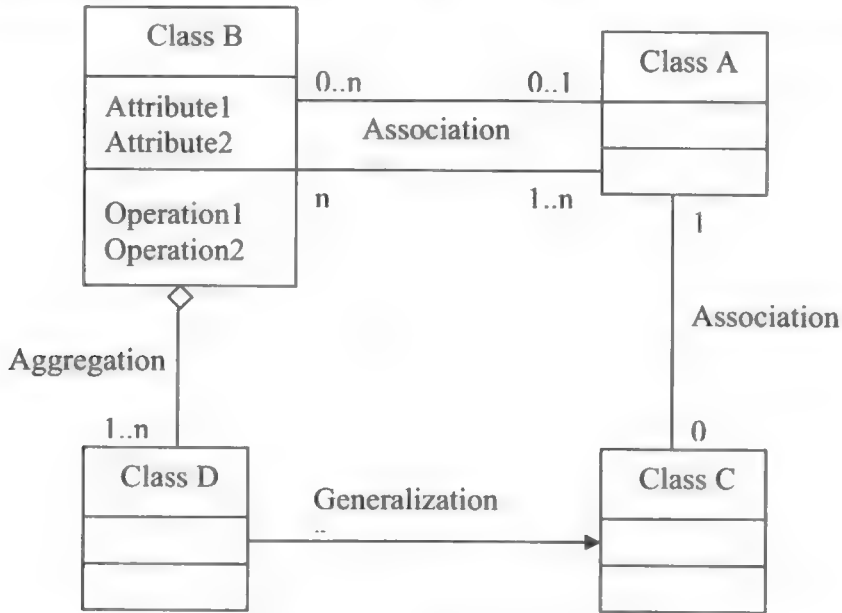
حالات الاستخدام للمجال فى مستودع النظام Repository. يوضح شكل (٥-٤) نموذجاً لمخطط حالات الاستخدام لأحد مجالات نظام الأعمال العام.

٥-٢-٣ مخطط الأصناف للمجال Domain Class Diagram

يمثل مخطط الأصناف للمجال المنظور التركيبى لأحد مجالات نظام الأعمال من خلال تمثيل أصناف المجال والعلاقات بينها فى شكل نموذج مرئى. يتضمن المخطط وصفاً لبعض الخصائص المهمة للأصناف خاصة التى تسهم فى عمليات الربط بين الأصناف وكذلك العمليات الرئيسية التى يقوم بها الصنف. ويوضح شكل (٥-٥) نموذجاً لمخطط الأصناف لأحد مجالات نظام الأعمال العام.



شكل (٥-٤) نموذج لمخطط حالات استخدام المجال



شكل (٥-٥) نموذج مخطط الأصناف للمجال

تدعم أدوات النمذجة باستخدام لغة UML إمكانية تسجيل الخصائص والعمليات التي لم تظهر في مخطط الأصناف بالإضافة إلى معلومات أخرى إضافية يتم تخزينها في مستودع نظام الأعمال.

يوضح الشكل إمكانية أن تكون علاقة الارتباط Association أحادية الاتجاه Unidirectional أي أنه لا توجد أي حالة من حالات الصنف A يمكن أن ترتبط بحالة من حالات الصنف C ولكن الارتباط يكون مسموحاً فقط من الصنف C إلى الصنف A.

٣-٥ حالة دراسية Case Study:

سنحاول في هذا الجزء تطبيق أسلوب التكرار مع الزيادة لتطوير التطبيق الخاص بإدارة البريد الإلكتروني (Email Management (EM وهو يمكن أن يكون جزءاً مهماً في العديد من أنظمة المعلومات. يمكننا أيضاً استخدام هذا التطبيق على أنه نظام تسويق عبر الإنترنت من خلال البريد الإلكتروني Email Marketing الذي يستهدف عدداً كبيراً من العملاء حول العالم ويمدهم بالمعلومات التسويقية عن منتجات معينة

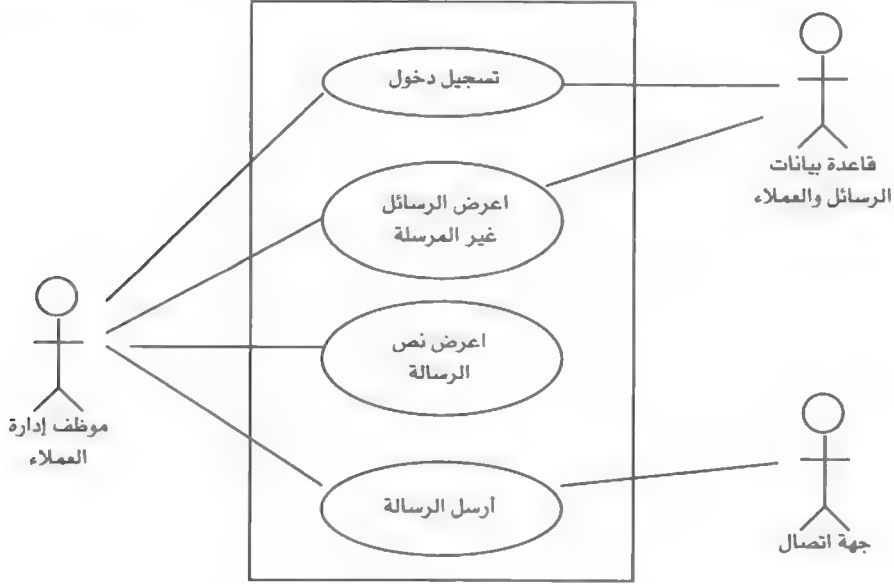
أو خدمات تقدم من خلال الإنترنت، وذلك باستخدام أسلوب الرسائل الإلكترونية عبر البريد الإلكتروني. يقوم النظام بتسجيل بيانات العملاء وعناوينهم الإلكترونية في قاعدة البيانات، ثم يقوم بتجهيز الرسائل الترويجية للمنتجات والخدمات، ويقوم بإرسالها لجميع العملاء. يقوم النظام أيضاً بعمليات تحديث دورية للبيانات المسجلة في قاعدة البيانات.

٥-٣-١ نموذج حالات الاستخدام Use Case Model،

سوف نفترض في هذه الحالة الدراسية أن الرسائل الترويجية قد تم تجهيزها وتسجيلها من قبل في قاعدة البيانات للنظام. وعلى هذا سيكون الهدف من الحالة الدراسية هو إرسال الرسائل المخزنة في قاعدة البيانات والاستعلام عنها. يمكننا توصيف متطلبات الحالة الدراسية من خلال مجموعة من حالات الاستخدام والفاعلين الذين يتعاملون معها. يوضح شكل (٥-٦) نموذج حالات الاستخدام لنظام إدارة البريد الإلكتروني EM.

يحتوى النموذج على أربع حالات استخدام هي:

- تسجيل دخول Login.
- أرسل الرسالة Send Message.
- اعرض نص الرسالة Display Message.
- اعرض الرسائل غير المرسلة View Unsent Messages.
- بالإضافة إلى مجموعة الفاعلين:
- موظف إدارة العملاء Customer Department Employee.
- جهة اتصال Contact.
- قاعدة بيانات الرسائل والعملاء Production Database.



شكل (٦-٥) نموذج حالات الاستخدام لنظام إدارة البريد الإلكتروني

٢-٣-٥ وثيقة حالة الاستخدام Use Case Document:

على الرغم من أهمية استخدام النماذج المرئية لوصف متطلبات النظام إلا أنها لا تكفي لمعرفة المتطلبات بشكل مفصل. يتم توصيف محتويات حالة الاستخدام بشكل نصي من خلال وثيقة تسمى «وثيقة حالة الاستخدام» وتكون مرفقة بنموذج حالات الاستخدام. يحدد فريق التطوير شكل قالب وثيقة حالات الاستخدام. يتكون القالب غالباً من عدة أجزاء بالإضافة إلى رأس الوثيقة Header الذي يتضمن معلومات عن مؤلف الوثيقة وتاريخ إنشائها وغيرها. يوضح شكل (٥-٧) أحد أشكال القوالب المستخدمة لوثيقة حالات الاستخدام. تبدأ الوثيقة بعنوان لحالة الاستخدام يليه وصف مختصر للحالة، ثم عرض لمجموعة الفاعلين للحالة. تحتوي الوثيقة أيضاً على وصف لأي شروط مسبقة يجب أن تتوافر قبل بدء حالة الاستخدام وكذلك وصف لأي شروط لاحقة يجب تنفيذها بعد انتهاء تنفيذ حالة الاستخدام. يحتوي الجزء الأساسي لحالة الاستخدام على مجموعة الخطوات الأساسية لتنفيذ الحالة وتسمى «التدفق الأساسي» وهو يمكن أن يحتوي على مجموعة من التدفقات الفرعية

بالإضافة إلى تدفقات استثنائية تصف مجموعة الأفعال التي يجب اتخاذها عند حدوث أخطاء في أثناء التنفيذ.

وفيما يلي عرض لمحتوى كل جزء من أجزاء وثيقة حالات الاستخدام لتطبيق إدارة البريد الإلكتروني EM.

وصف مختصر لحالة الاستخدام:

تهدف حالة الاستخدام إلى عرض محتويات الرسائل الموجودة بقاعدة البيانات وإرسالها إلى العملاء المقيدين بالنظام.

الفاعلون:

موظف إدارة العملاء.

قاعدة بيانات الرسائل والعملاء.

جهة اتصال.

شروط مسبقة:

تخزين الرسائل المطلوب إرسالها في قاعدة البيانات.

منح الموظف المختص بإدارة العملاء صلاحية استخدام قاعدة البيانات.

شروط لاحقة:

التعديل في معلومات قاعدة البيانات لتعكس حالة الرسائل التي تم إرسالها بنجاح.

الاستمرار في الاتصال بقاعدة البيانات حتى يتم إرسال باقى الرسائل.

التدقق الأساسى:

يطلب النظام معلومات المستخدم وكلمة المرور.

بعد نجاح عملية الدخول، يستعرض النظام قائمة الاختيارات المتاحة للمستخدم وهى:

- ١- استعراض محتويات الرسالة.
- ٢- عرض قائمة الرسائل التي لم ترسل.
- ٣- إرسال رسالة.
- ٤- خروج.

رقم النسخة:	تاريخ الإنشاء:
المؤلف:	
<p>عنوان حالة الاستخدام Use Case Title</p> <p>وصف مختصر لحالة الاستخدام Brief Description:</p> <p>الفاعلون Actors:</p> <p>شروط مسبقة Preconditions:</p> <p>شروط لاحقة Postconditions:</p> <p>التدفق الأساسي Basic Flow:</p> <p>التدفقات الفرعية Subflows:</p> <p>تدفقات استثنائية Exception Flows:</p>	

شكل (٧-٥) أحد قوالب وثيقة حالات الاستخدام

التدفقات الفرعية:

يحتوي التدفق الأساسي على مجموعة التدفقات الفرعية التالية:

S١: استعراض محتويات الرسالة:

يقوم الموظف بإدخال رقم الرسالة المطلوبة.

يتم عرض محتوى الرسالة.

يتم عرض قائمة الاختيارات مرة أخرى.

S2: عرض قائمة الرسائل التي لم ترسل:

عرض رسالة تحتوي على عدد الرسائل التي لم ترسل بعد.

عرض المحتويات العامة عن كل رسالة (رقمها، التاريخ، الموضوع، العميل).

يتم عرض قائمة الاختيارات مرة أخرى.

S3: إرسال الرسالة.

يقوم الموظف بإدخال رقم الرسالة المطلوب إرسالها.

تنفيذ عملية إرسال الرسالة.

تعديل بيانات الرسائل في قاعدة البيانات.

يتم عرض قائمة الاختيارات مرة أخرى.

تدفقات استثنائية:

يحتوي النظام على التدفقات الاستثنائية التالية:

E1: اسم مستخدم أو كلمة مرور غير صحيحة:

هي رسالة تظهر للمستخدم عند إدخال اسم مستخدم User Name أو كلمة مرور Password غير صحيحة.

يسمح النظام بأن يقوم المستخدم بإعادة إدخال بيانات المستخدم وكلمة المرور أو الخروج.

يسمح النظام بثلاث محاولات فقط لإدخال البيانات بشكل صحيح.

E2: اختيار غير صحيح:

هي رسالة تظهر للمستخدم عند إدخال رقم لا يتوافق مع أرقام الاختيارات المعروضة في التدفق الأساسي للنظام.

يتم عرض قائمة الاختيارات مرة أخرى.

E٣: عدد كبير جداً من الرسائل:

هى رسالة يمكن أن تظهر فى أثناء تنفيذ التدفق الفرعى S١ (عرض قائمة الرسائل التى لم ترسل) إذا كان عدد الرسائل غير المرسله كبيراً يتعدى الرقم المحدد من قبل لعدد الرسائل التى يمكن عرضها.

يقوم النظام بعرض عدد الرسائل المسموح بها.

يتم عرض رسالة تفيد بوجود رسائل أخرى لم ترسل وموجودة بقاعدة البيانات.

E٤: رسالة لم يتم إرسالها:

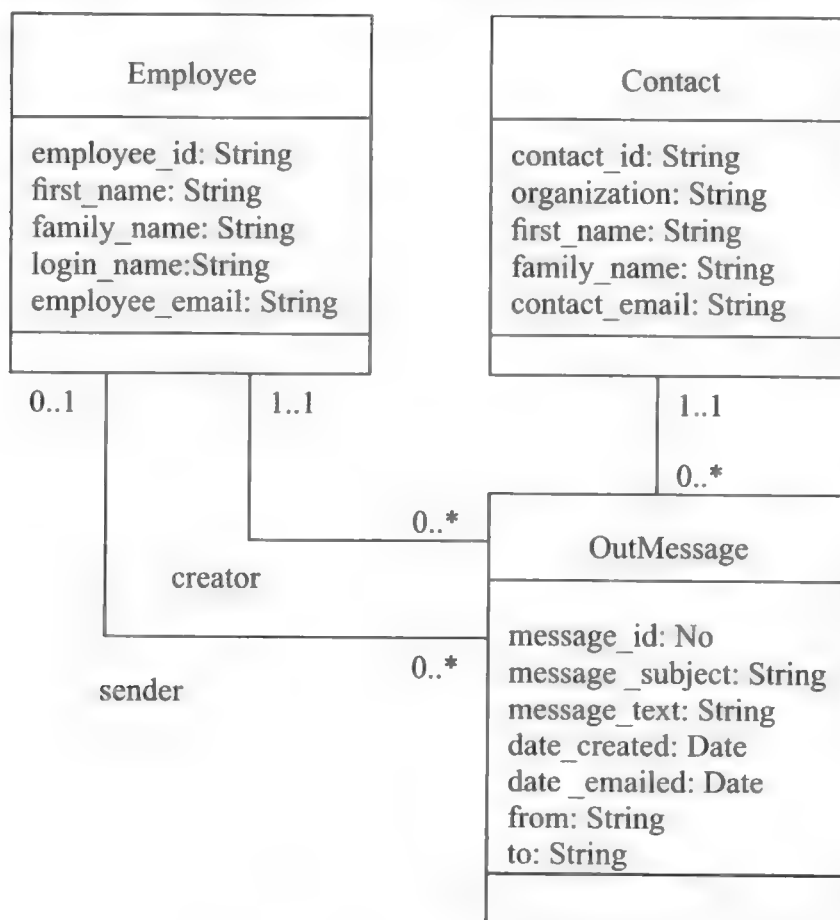
هى رسالة يمكن ظهورها فى أثناء تنفيذ التدفق الفرعى S٣ (إرسال رسالة) فى حالة عدم إتمام عملية إرسال الرسالة بنجاح.

٣-٣-٥ الأصناف Classes:

يحتوى نظام إدارة البريد الإلكتروني على ثلاثة أصناف هى:

Employee, Contact, and Outmessage

يوضح شكل (٥-٨) مخطط الأصناف للنظام. يحتوى المخطط على الأصناف والعلاقات بينها مع توضيح اسم العلاقة وعدد الكائنات المشاركة فيها من كل صنف. وقد تم عرض مجموعة الخصائص المهمة لكل صنف ولم يتم عرض العمليات فى هذه المرحلة الأولية لنموذج تحليل المتطلبات. نلاحظ أيضاً من المخطط وجود أكثر من علاقة ربط بين الصنفين Employee, Outmessage. العلاقة الأولى تسمى «Creator» وهى تصف العلاقة بين الرسالة والموظف الذى قام بإنشائها وتخزينها فى قاعدة البيانات. أما العلاقة الثانية فتسمى «Sender»، وهى تصف العلاقة بين الرسالة والموظف الذى سيقوم بإرسالها.



شكل (٨-٥) مخطط الأصناف للتطبيق EM

الفصل السادس

التصميم المعماري للنظام

Architectural System Design

يعتبر التصميم المعماري للنظام البرمجي هو الأساس الذي تبنى عليه جميع الحلول والتصميمات الأخرى داخل النظام. ولذلك فإنه لا يمكن بناء أى نظام برمجيات بدون تصميم معمارى مسبق للنظام. يهتم التصميم المعماري للبرمجيات بعمليات تنظيم الوحدات البرمجية مثل: الأصناف Classes، والمكونات Components، والحزم Packages وكذلك توضيح الاتصالات الداخلية بين الوحدات.

٦-١ الطبقات المعمارية وإدارة التبعية Architectural Layers and Dependency Management

يتميز التصميم المعماري للنظم البرمجية باعتماده أسلوب هيكل النظام فى شكل طبقات من الوحدات مع تنظيم الاتصالات بين الوحدات. يؤدى هذا الأسلوب إلى تقليل درجة التعقيد فى الهيكل المعماري للنظام وتحسين درجة الفهم للتبعيات بين الوحدات. ومن الفوائد المهمة أيضاً لاستخدام التصميم المعماري ضمان استخدام أسلوب الوحدات البرمجية المستقلة والتفاعل فيما بينها وهو يتوافق تماماً مع أسلوب الكائنات الموجهة Object-Oriented لإنتاج البرمجيات.

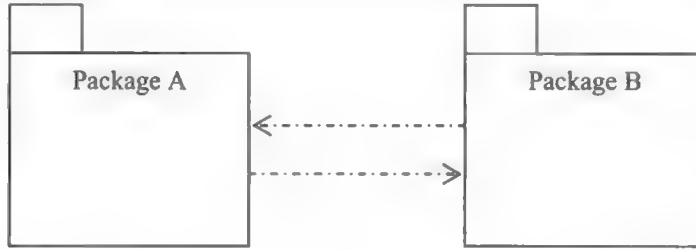
٦-١-١ الوحدات البرمجية المعمارية Architectural Modules

يعتبر الصنف Class من الوحدات البرمجية التى تستخدم على نطاق واسع خلال مراحل التطوير المختلفة لمشروع البرمجيات، فهو يستخدم لوصف مجموعة الفاعلين Actors المتعاملين مع النظام وكذلك مجموعة الكائنات البرمجية للنظام البرمجي مثل: كائنات التعامل مع قواعد البيانات، وكائنات واجهات المستخدم، وكائنات إجراء العمليات، وغيرها. ويستخدم الصنف فى المستويات المختلفة مثل: الوحدة البرمجية أو التطبيق أو النظام العام. ويستخدم الصنف أيضاً فى جميع مراحل التطوير مثل:

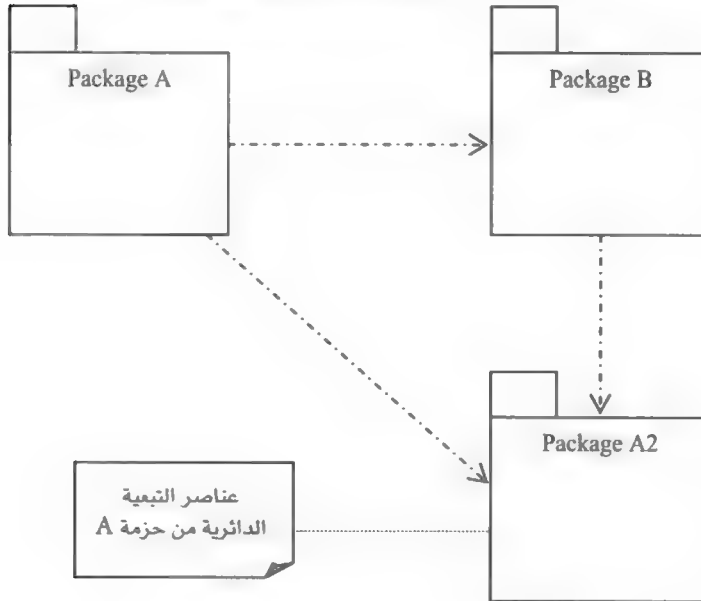
التحليل والتصميم والبرمجة. يتم تجميع الأصناف حسب الإطار المعماري المناسب لتطوير المشروع في شكل مجموعة من الحزم Packages يتم تنظيمها أيضاً في شكل معماري يصف العلاقات بينها. يساهم التصميم المعماري في تنظيم عملية التبعية بين الوحدات، فيمكن اعتبار إحدى الوحدات A معتمدة على وحدة أخرى B إذا كان التغيير في الوحدة B يؤدي إلى تغيير في الوحدة A. وكما ذكرنا سابقاً فإن لغة النمذجة الموحدة UML تدعم أسلوب تصميم الحزم التي يتم تطبيقها بشكل مماثل في أثناء البرمجة باستخدام لغات البرمجة الحديثة المبنية على أسلوب الكائنات الموجهة مثل لغة جافا Java. تحتوي الحزمة على مجموعة من الأصناف التي يتم استدعاؤها مرة واحدة عند استدعاء الحزمة، وكذلك يمكن حذفها كاملة من التطبيق. يمكن أن تحتوي الحزمة على حزم أخرى بداخلها. وعلى هذا فإن أي صنف يمكن أن يكون عضواً في حزمة واحدة فقط ولكن يمكن تصديره إلى حزم أخرى.

٦-١-٢ تبعيات الحزم البرمجية Packages Dependencies،

تحتاج عملية تنظيم الحزم بشكل معماري إلى إدارة لعملية التبعية بين الحزم. ومن المعروف أنه لزيادة كفاءة عملية إعادة استخدام الوحدات البرمجية، فإنه يجب علينا أن نقلل من الاعتمادية بين الوحدات بقدر الإمكان وكذلك التخلص من التبعية غير الضرورية بين الوحدات من التصميم المعماري للنظام. ومن التبعية غير المرغوب فيها وتسبب مشكلات كبيرة في أثناء التطبيق التبعية الدائرية أو المغلقة Circular Dependencies. يوضح شكل (٦-١) مثالاً لتبعية دائرية بين حزمتين، حيث تعتمد الحزمة A على الحزمة B وكذلك تعتمد الحزمة B على الحزمة A. ويؤدي ذلك إلى مشكلات كبيرة في أثناء التطبيق والصيانة وإعادة الاستخدام. يمكن التغلب على تلك المشكلة بإضافة حزمة ثالثة A٢ تحتوي على عناصر من الحزمة A التي تعتمد عليها الحزمة B، كما هو موضح بشكل (٦-٢). وعلى هذا فلم تعد الحزمة B تعتمد على الحزمة A ولكنها أصبحت تعتمد على الحزمة A٢ فقط. وبالتبع إن الحزمة A تعتمد على الحزمة A٢، وبذلك تم إزالة التبعية الدائرية.



شكل (١-٦) التبعيات الدائرية بين الحزم

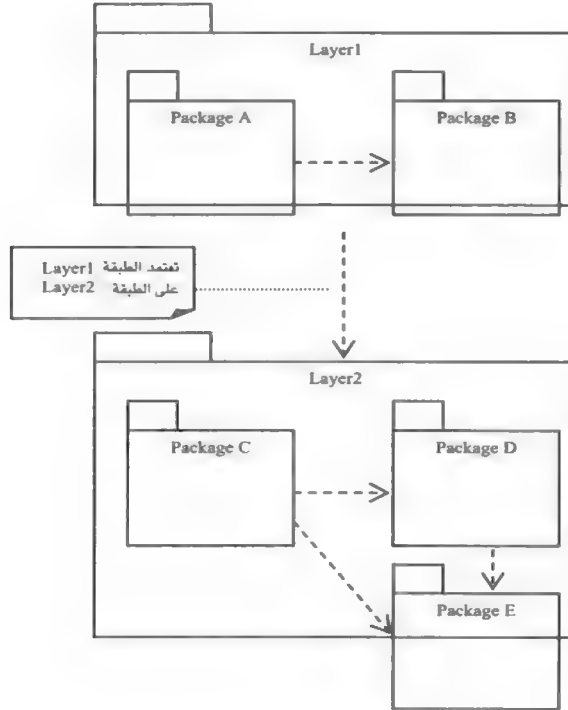


شكل (٢-٦) إزالة التبعيات الدائرية

٣-١-٦ تبعيات الطبقة Layer Dependencies

يمكننا تجميع الحزم وتنظيمها في شكل طبقات هرمية Hierarchical Layers وذلك في محاولة لدعم عملية وصف معمارية النظام. ومن المعروف أن الطبقات تأخذ الشكل الرأسى، إذ تحتوى كل طبقة على مجموعة من الحزم مما يؤدي إلى التصميم الهرمى للحزم كما هو موضح بالشكل (٣-٦). يهدف التصميم الجيد لمعمارية الطبقات إلى ثلاثة أهداف هي:

- ١- أن تساعد هرمية الطبقات على التكامل مع باقى مكونات النظام.
- ٢- أن تؤدي هرمية الطبقات إلى تقليل التبعية بين الحزم.
- ٣- أن تسهم هرمية الطبقات فى إمكانية الحصول على إطار عمل مستقر لعملية تطوير النظام.



شكل (٦-٣) تبعيات الطبقة

٦-١-٤ تبعيات الصنف Class Dependencies:

يجب التخلص من التبعية الدائرية الموجودة بين الطبقات وذلك لضمان الحصول على تصميم معمارى مستقر للنظام. وإمكانية تحقيق ذلك يجب معرفة الأسباب التى تؤدي إلى وجود التبعية الدائرية. فمن المعروف أن التبعية بين الطبقات تنشأ نتيجة وجود تبعيات بين الحزم فى الطبقات، وأن التبعية بين الحزم تنشأ نتيجة وجود تبعيات بين الأصناف داخل الحزم. يوضح شكل (٦-٤) هذا المفهوم، إذ نجد أن الطبقة Layer1 تعتمد على الطبقة

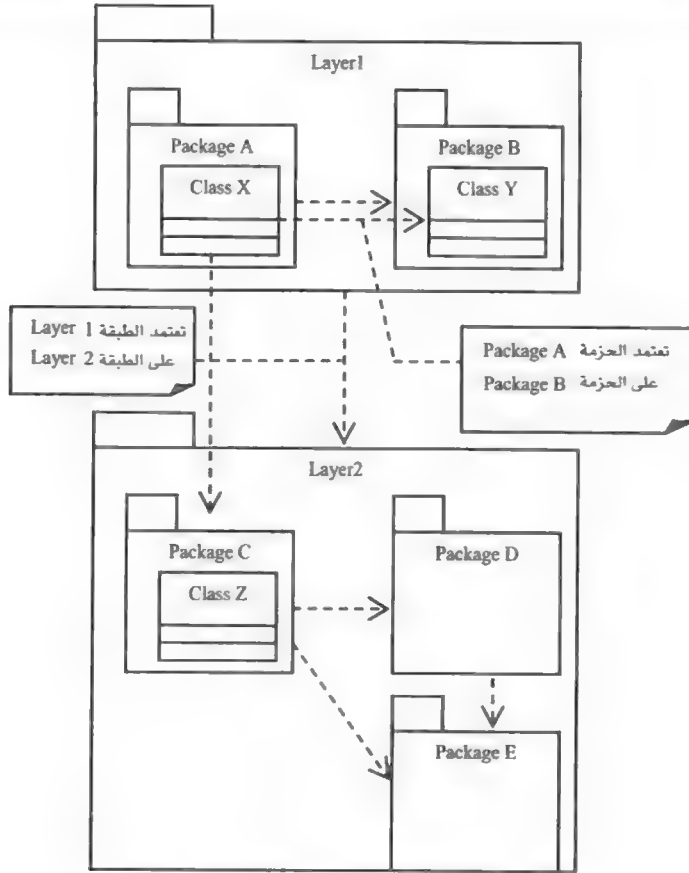
Layer₂ لأن هناك صنف Class X فى الطبقة Layer₁ يعتمد على الصنف Class Z فى الطبقة Layer₂. ونجد أيضاً أن الحزمة Package A تعتمد على الحزمة Package B لأن هناك صنف Class X فى الحزمة Package A يعتمد على الصنف Class Y فى الحزمة Package B. وعلى هذا فإنه يجب التركيز على إزالة التبعية الدائرية التى يمكن أن تنشأ بين الأصناف وذلك يؤدى إلى إزالة أى تبعيات دائرية بين الحزم أو بين الطبقات.

٥-١-٦ تبعية التوريث Inheritance Dependencies

تعتبر الاعتمادية بين الأصناف الناتجة عن خاصية التوريث من التبعية التى يمكن أن تؤدى إلى مشكلات كبيرة. فمن المعروف أنه فى حالة تطبيق أسلوب التوريث نجد أن الصنف الفرعى sub class يرث كل خصائص وطرق الصنف العلوى super class مع إمكانية قيامه بتغيير أو استبدال بعض هذه الخصائص والطرق وإضافة خصائص وطرق جديدة له. ويؤدى ذلك إلى تغيير فى سلوك النظام فى أثناء التشغيل اعتماداً على سلوك الكائنات الفرعية المستخدمة فى أثناء تنفيذ عملية التوريث فعلياً. يمكننا تصنيف تبعيات التوريث إلى نوعين هما:

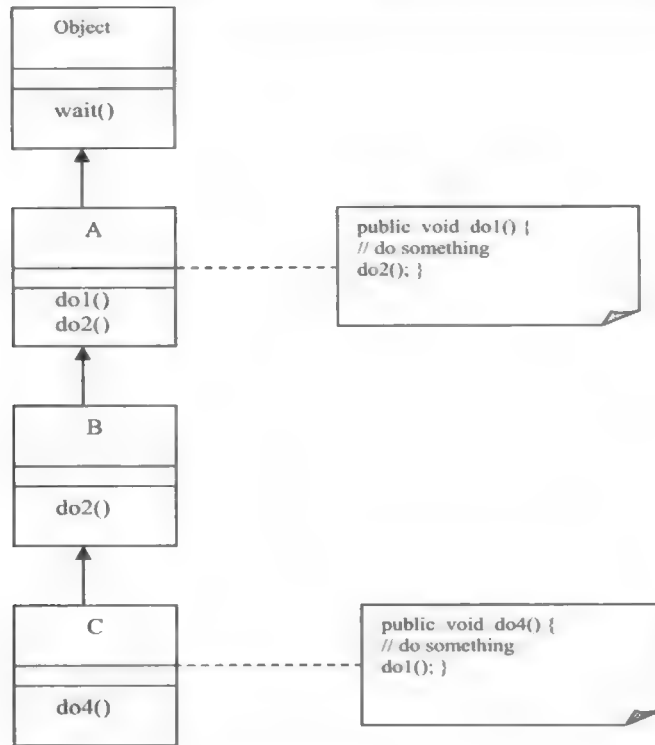
- تبعيات بين الأصناف فى شجرة توريث فى أثناء وقت الترجمة compile time.
- تبعيات بين الكائنات فى وقت التنفيذ run time وهى تستخدم خدمات تقدمها أصناف موجودة فى شجرة توريث.

يوضح شكل (٥-٦) مثلاً على التبعية بين الأصناف فى شجرة توريث فى أثناء وقت الترجمة. تؤدى علاقة التعميم generalization relationship إلى وجود تبعيات فى اتجاه الأسهم، حيث نجد أن الصنف C يعتمد على الصنف B وهو يعتمد بدوره على الصنف A الذى يعتمد على الصنف العام object. نلاحظ أيضاً من الشكل وجود تبعيات فى وقت التنفيذ من خلال الطريقة الموجودة فى الصنف B الذى يمكن أن يستبدل الطريقة نفسها do₂() الموجودة فى الصنف A. ويمكن أيضاً أن تنشأ تبعية دائرية نتيجة للتوريث بين الأصناف. ففى حالة قيام الطريقة do₁() فى الصنف A باستدعاء الطريقة do₂() فى الصنف نفسه ويمكن استبدالها بالطريقة do₂() الموجودة بالصنف B، فإن ذلك يؤدى إلى وجود تبعية دائرية يصعب التحكم فيها وتؤدى إلى مشكلات كبيرة.

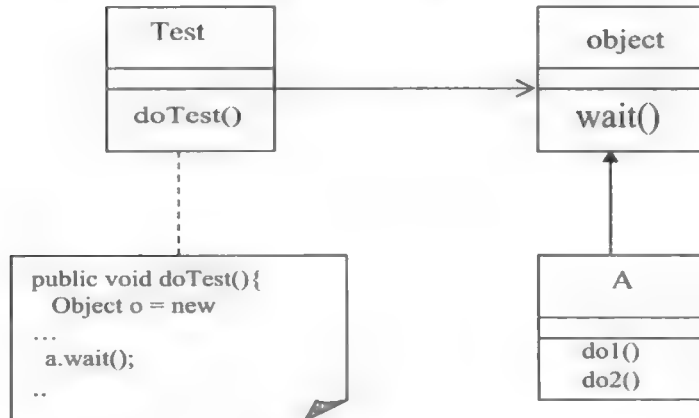


شكل (٦-٤) تبعيات الصنف

يوضح شكل (٦-٦) مثالاً على تبعيات التوريث في أثناء التنفيذ run time. نلاحظ من المثال أن التبعية تحدث نتيجة التوريث عندما يستدعي الكائن test الطريقة wait() من خلال الكائن A في وقت التنفيذ، إذ إن الكائن A يرتبط بالكائن object بالتوريث ويمكنه أن يغير من سلوك الطريقة wait() في أثناء التنفيذ. ويتطلب ذلك الحرص الشديد من فريق التصميم ومراعاة تجنب حدوث تلك الأنواع من التبعية الدائرية، بالإضافة إلى الحرص عند استخدام أسلوب الاستبدال للطرق method overriding في وجود التبعية بين الأصناف.



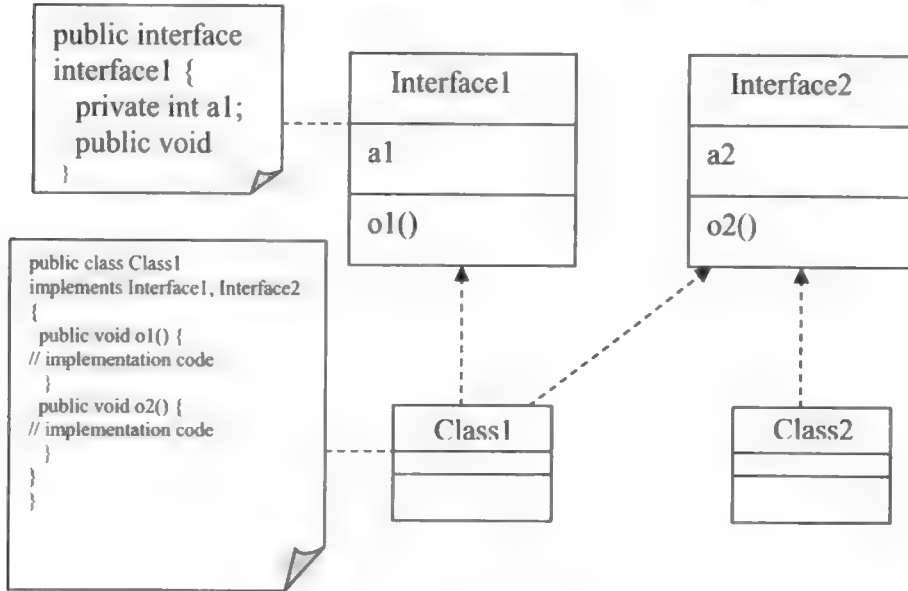
شكل (٥-٦) تبعيات التوريث في أثناء الترجمة



شكل (٦-٦) تبعيات التوريث في أثناء التنفيذ

٦-١-٧ الواجهات Interfaces

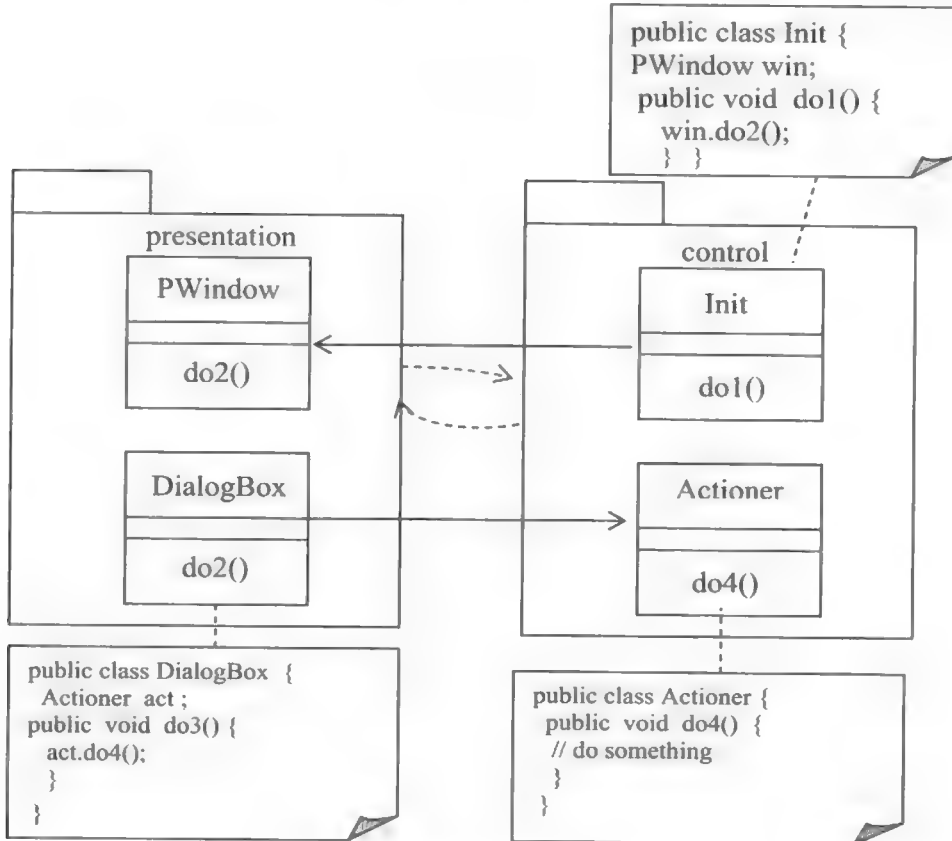
كما ذكرنا سابقاً فإن الواجهة هي: مجرد تعريف لمجموعة من الخصائص والطرق التي لا يمكن إنشاء كائنات منها بطريقة مباشرة ولكن تحتاج إلى صنف آخر يقوم بتطبيقها أو تحقيقها وذلك من خلال توصيف الأكواد التفصيلية للطرق الموجودة في صنف الواجهة أي أنها تقوم بتحقيقها فعلياً. يعتبر ذلك نوعاً من أنواع التبعية التي تسمى تبعية التطبيق Implementation Dependency وهي توجد بين صنف الواجهة والصنف الذي يقوم بتطبيقه. يمكن أن يقوم صنف واحد بتحقيق أكثر من صنف واجهة، ويمكن أن يقوم أكثر من صنف واحد بتحقيق صنف الواجهة. يوضح شكل (٦-٨) مثالاً على تبعية التطبيق من خلال علاقة التبعية بين صنف الواجهة وصنف التحقيق. نلاحظ من الشكل أن اتجاه العلاقة يكون ناحية صنف الواجهة. ونلاحظ أيضاً أن الصنف Class1 يقوم بتحقيق صنفين الواجهة Interface1 و Interface2 وكذلك نجد أن كلا الصنفين Class1 و Class2 يمكنهما تحقيق صنف الواجهة Interface2.



شكل (٦-٨) تبعية التطبيق

ومع ذلك فإن من مميزات استخدام صنف الواجهة هو إزالة التبعية الدائرية Circular Dependency بين الأصناف. ولتوضيح ذلك نستعين بالمثال الموضح بالشكل (٦-٩) وهو يحتوى على تبعية دائرية بين الحزمتين Presentation و Control.

ولحل هذه المشكلة وكسر التبعية الدائرية يتم استخدام صنف الواجهة في إحدى الحزم كما هو موضح بالشكل (٦-١٠). يقوم صنف الواجهة Presenter بتعريف الطريقة do2() التي يستخدمها الصنف Init والموجود معه في الحزمة نفسها Control ويتم تطبيق صنف الواجهة من خلال الصنف PWindow الموجود في الحزمة Presentation. وعلى هذا فإن الصنف Init سوف يستخدم صنف الواجهة الذي يتم تطبيقه من خلال الصنف PWindow.



شكل (٦-٩) مثال على التبعية الدائرية

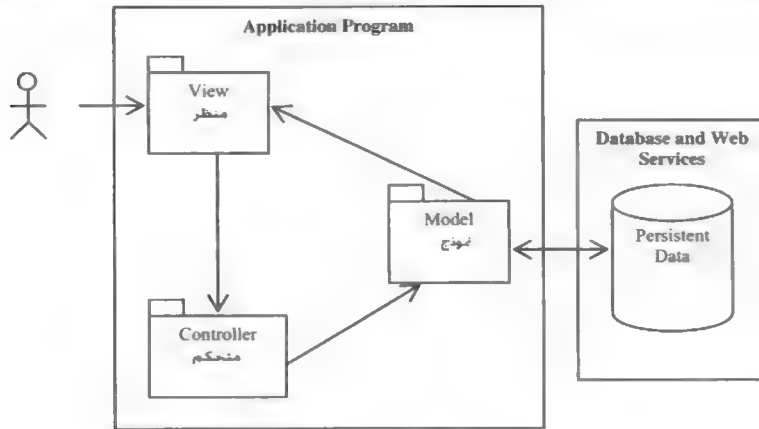
٦-٢ أطر معمارية Architectural Frameworks

تهتم النظم المعتمدة على أسلوب الكائنات الموجهة ببناء أطر معمارية تستخدم فى أثناء عمليات التصميم والبرمجة التى تعزز عملية إعادة الاستخدام Reuse فى مرحلتى التصميم والبرمجة. سنقدم الآن عرضاً لبعض تلك الأطر المعمارية التى تدعم عملية إعادة الاستخدام، إذ يقوم المصمم والمبرمج بتخصيص هذه الأطر والإضافة إليها لتناسب احتياجاته وذلك فى حدود الهيكل التصميمى للإطار المعماري المستخدم.

٦-٢-١ إطار نموذج- منظر - متحكم (MVC) Model-View-Controller

يعتبر إطار MVC أحد أطر التصميم المبنية على الكائنات وقدمت من خلال بيئة التطوير Smalltalk-80. يقوم هذا الإطار على فكرة الفصل بين المفاهيم للتصميم المبنى على الكائنات وذلك بتقسيم كائنات التطبيق إلى ثلاث مجموعات ترتبط بثلاثة أصناف مجردة فى شكل الإطار Controller و View و Model من خلال علاقة التعميم Generalization، إذ ترث منها وتضيف إليها خصائصها المميزة. يتم تمثيل كائنات البيانات Data Objects من خلال كائنات النموذج للإطار Model Objects. يتم إشعار كل من كائنات المنظر View Objects وكائنات المتحكم Controller Objects بأى تغييرات تحدث فى كائنات النموذج باستخدام أسلوب معالجة الأحداث، إذ يعتبر النموذج هو مؤسس الحدث Event Publisher وتستجيب كل من كائنات المنظر والمتحكم لهذا الحدث. يتم تمثيل كائنات واجهة المستخدم GUI Objects من خلال كائنات المنظر للإطار View Objects وهو يصف حالة النموذج بشكل رسومي يناسب المستخدم. يستجيب المنظر للتغييرات التى تحدث فى كائنات النموذج ويقوم بتحديث البيانات التى تعرضها للمستخدم. ومن الطبيعى أن يرافق كل كائن منظر كائن آخر متحكم لتمثيل الأحداث.

يتم تمثيل الأحداث من خلال كائنات المتحكم التى تستجيب لطلبات المستخدم فى أثناء التفاعل مع النظام من خلال كائنات المنظر. وتعتبر كائنات المتحكم بمنزلة الوسيط بين كائنات النموذج وكائنات المنظر، إذ تقوم بتفسير الحدث مثل النقر على الفأرة أو مفاتيح الإدخال وتحويلها إلى أفعال تطبق على كائنات النموذج. يوضح شكل (٦-١١) إطار MVC والاتصال بين الكائنات الثلاثة، إذ يقوم كائن المنظر باستقبال الحدث من مستخدم النظام ويقوم بتمريره إلى الكائن المتحكم لتفسيره وتحويله إلى الفعل المناسب ليتم تطبيقه على كائنات النموذج.



شكل (١١-٦) إطار MVC

يحقق استخدام هذا الإطار الثلاثي العديد من المميزات مثل:

- السماح بالتطوير المنفصل لكل من واجهة المستخدم والنموذج الذي يتضمن كلاً من بيانات وعمليات كائنات نظام الأعمال.
- إمكانية استخدام أكثر من واجهة مستخدم بدون إجراء تعديلات على النموذج.
- إمكانية إعادة تصميم النموذج مع الحفاظ على واجهة المستخدم بدون تغيير.
- إمكانية تغيير طريقة استجابة واجهة المستخدم للأحداث بدون تغيير واجهة المستخدم.

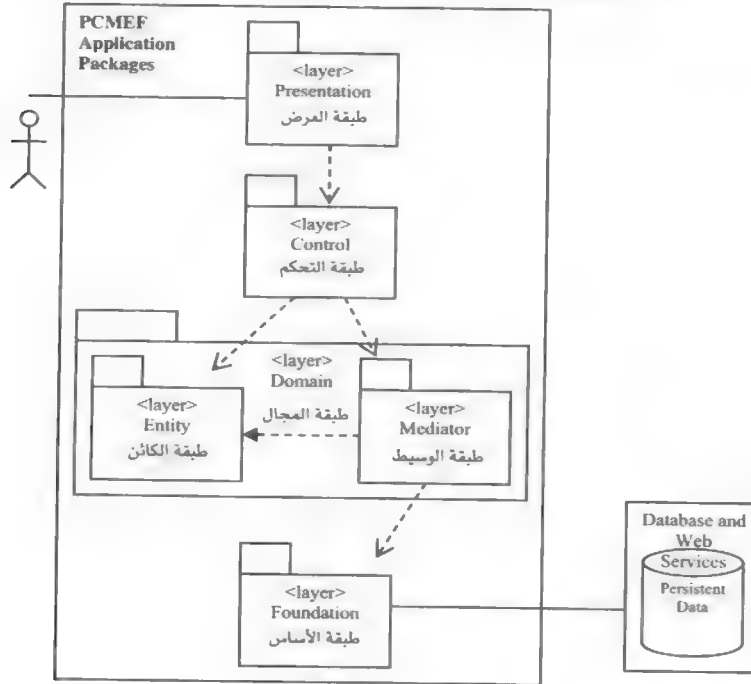
٢-٢-٦ إطار عرض-تحكم-وسيط - كائن - أساس - Presentation-Control-Mediator-Entity-Foundation (PCMEF)

نقدم الآن إطار عمل آخر يسمى PCMEF وهو يتكون من مجموعة من الطبقات ويأخذ الشكل المعماري الهرمي. يتكون الإطار من أربع طبقات رأسية هي:

- العرض Presentation.
- التحكم Control.
- المجال Domain.
- الأساس Foundation.

وتتكون طبقة المجال من حزمتين هما الكائن Entity والوسيط Mediator. يوضح شكل (١٢-٦) التنظيم الهرمي لطبقات الإطار. نلاحظ من الشكل وجود تبعيات بين الطبقات من أعلى إلى أسفل، حيث تعتمد طبقة العرض على طبقة التحكم التي تعتمد بدورها على طبقة المجال، وتعتمد حزمة الوسيط على حزمة الكائن وعلى طبقة الأساس.

تحتوي طبقة العرض Presentation على كائنات واجهة المستخدم GUI Objects تستخدم في تعامل المستخدم مع النظام. وتحتوي طبقة التحكم Control على الكائنات المسؤولة عن معالجة تفاعلات المستخدم مع النظام من خلال مجموعة الخدمات الوظيفية التي يقدمها النظام. وتحتوي حزمة الكائن Entity على كائنات النظام التي سيتم تخزينها في قاعدة البيانات أو التي يتم استرجاعها من قاعدة البيانات. تقوم الحزمة Mediator بدور الوسيط أو قناة اتصال بين كائنات حزمة الكائن Entity وكائنات طبقة الأساس Foundation. تقوم طبقة الأساس بجميع العمليات المسؤولة عن الاتصال بقواعد البيانات وإدارة العمليات التي تتم على البيانات مثل: التسجيل والاستعلام والاسترجاع وغيرها.



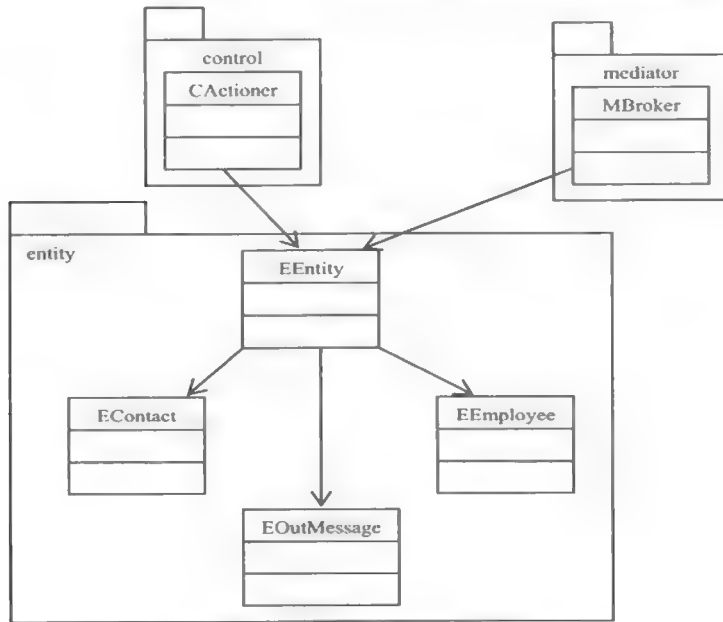
شكل (١٢-٦) إطار PCMEF

٣-٦ أنماط معمارية Architectural Patterns:

نستعرض في هذا الجزء بعض الأنماط المعمارية التي تم استخدامها في تصميم الأطر المعمارية مثل إطار PCMEF. تستخدم هذه الأنماط لحل بعض مشكلات التصميم مثل: التحكم في التبعيات وإعادة استخدام الوظائف واستخدام أساليب التجريد Abstraction وإخفاء المعلومات Information Hiding (Gamma et al, 1995).

١-٣-٦ الواجهة Facade:

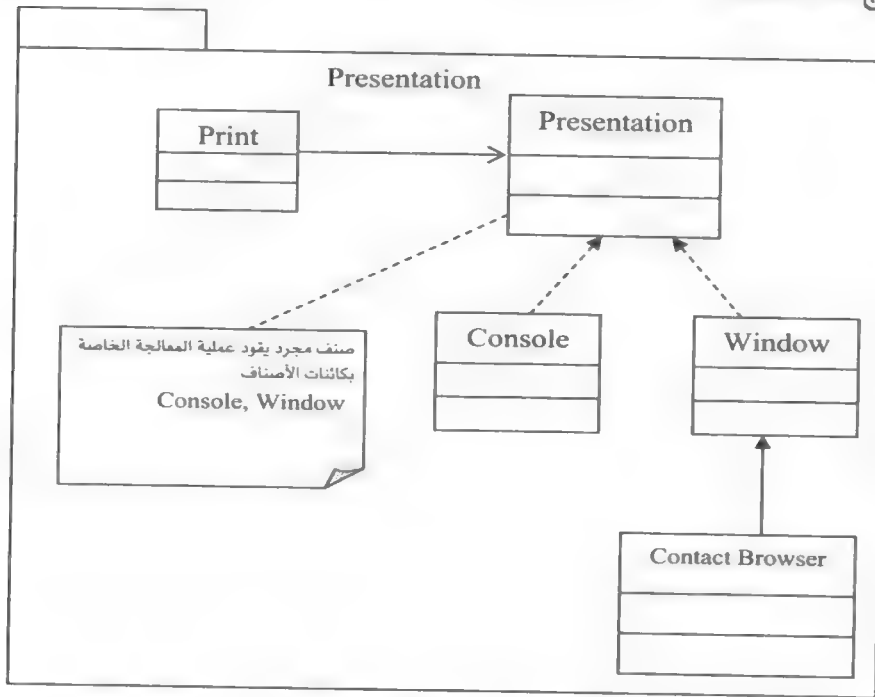
يعتبر الغرض الأساسي من استخدام نمط الواجهة هو تقليل عدد الاتصالات بين الحزم لأقل ما يمكن مما يقلل من التبعيات بين الحزم. يتم استخدام صنف الواجهة في المستوى الأعلى داخل الحزمة ويكون هو نقطة الدخول الرئيسية للحزمة عند التعامل مع الحزم الأخرى. تقوم الواجهة بتوجيه طلب الخدمة إلى الكائن المناسب لتنفيذها داخل الحزمة مما يؤدي إلى تقليل عدد مسارات الاتصال بين كائنات الحزمة والحزم الأخرى. يوضح شكل (٦-١٣) كيفية اتصال كل من حزمة Mediator وحزمة Control مع حزمة Entity من خلال صنف الواجهة EEntity الذي يقوم بعملية توجيه الاتصال إلى الكائن المناسب داخل الحزمة.



شكل (٦-١٣) نمط الواجهة

٢-٣-٦ المصنع المجرد Abstract Factory:

يقدم هذا النمط أسلوباً آخر لتقليل التبعيات بين الحزم من خلال واجهة مجردة تقوم بإنتاج مجموعة من الكائنات المختلفة خلف الواجهة. يؤدي هذا الأسلوب إلى إمكانية أن يتصرف التطبيق بأسلوب مختلف عندما يتعامل مع الكائنات المولدة حسب نوع الخدمة المطلوبة. يوضح شكل (٦-١٤) مثالاً لاستخدام نمط المصنع المجرد، حيث يمكن للتطبيق استخدام كائنات لوحة المفاتيح PConsole أو كائنات النوافذ PWindow وذلك من خلال الصنف المجرد IPPresentation الذي يقوم بإنتاج تلك الكائنات.

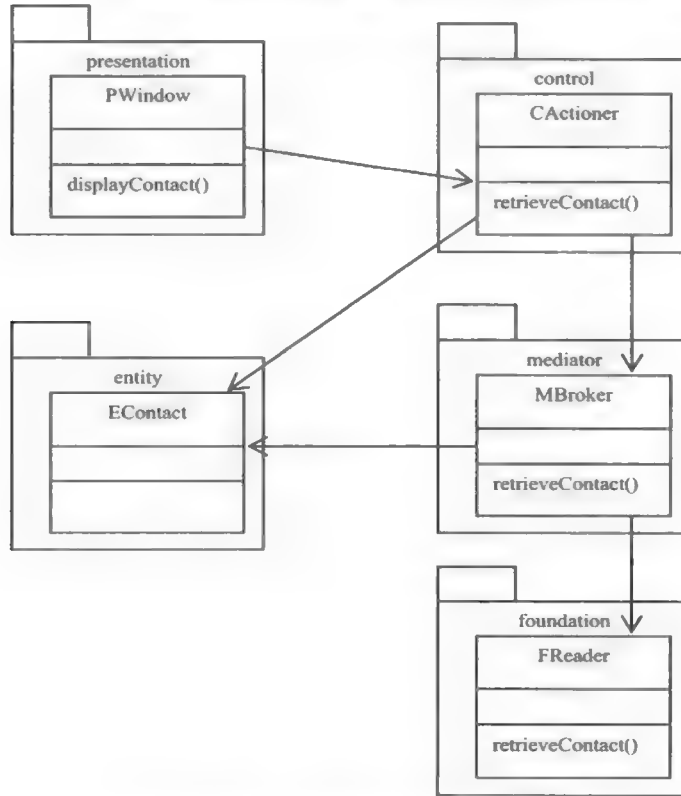


شكل (٦-١٤) نمط المصنع المجرد

٣-٣-٦ سلسلة المسؤوليات Chain of Responsibilities:

يعتبر الغرض من استخدام نمط سلسلة المسؤوليات هو تجنب الاقتران المباشر بين الكائن المرسل لطلب الخدمة مع الكائن المستقبل لها وإعطاء الفرصة لأكثر

من كائن للتعامل مع الطلب. يوضح شكل (٦-١٥) مثلاً لاستخدام هذا النمط، حيث يقوم الكائن PWindow بطلب خدمة () display Contact بأن يرسل رسالة للكائن CActioner الذي يمرر بدوره هذا الطلب إلى الكائن EContact إذا كان موجوداً في الذاكرة من قبل، أو يمرر الطلب إلى الكائن الوسيط MBroker الذي يقوم بتفويض الكائن FReader لاسترجاع الكائن المطلوب من قاعدة البيانات.

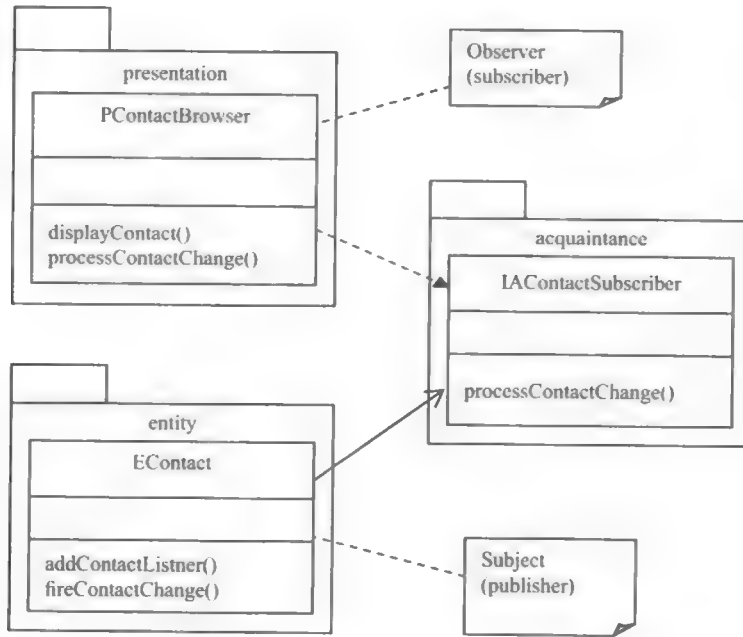


شكل (٦-١٥) نمط سلسلة المسؤوليات

٦-٣-٤ نمط المراقب Observer:

يعتبر الغرض من استخدام نمط المراقب هو تأسيس علاقة تبعية واحد - إلى - متعدد بين الكائنات، بحيث إذا حدث تغيير في الكائن فإنه يتم تحديث الكائنات التابعة له آلياً. يسمى الكائن الأول كائن «Subject» أو «Publisher» وتسمى الكائنات التابعة له كائنات

«Observers» أو «Subscribers». يتم إعلام الكائنات التابعة بأية تغييرات تحدث في الكائن الأول وتقوم بإجراء التعديلات الضرورية لتلائم حالتها مع حالة الكائن الأول. يؤدي هذا الأسلوب إلى تقليل درجة الاقتران بين الكائنات الذي يحدث نتيجة وجودها داخل حزم مختلفة في شكل طبقات داخل إطار معماري مثل PCMEF. تتم عملية الإعلام عن التغييرات في شكل سلسلة من أسفل إلى أعلى داخل الإطار. يوضح شكل (٦-١٦) مثالا لاستخدام نمط المراقب، إذ يقوم الكائن EContact بإعلام الكائن PContactBrowser بالتغييرات التي حدثت له وذلك من خلال كائن وسيط نظراً لوجود الكائنات في طبقات مختلفة.



شكل (٦-١٦) نمط المراقب

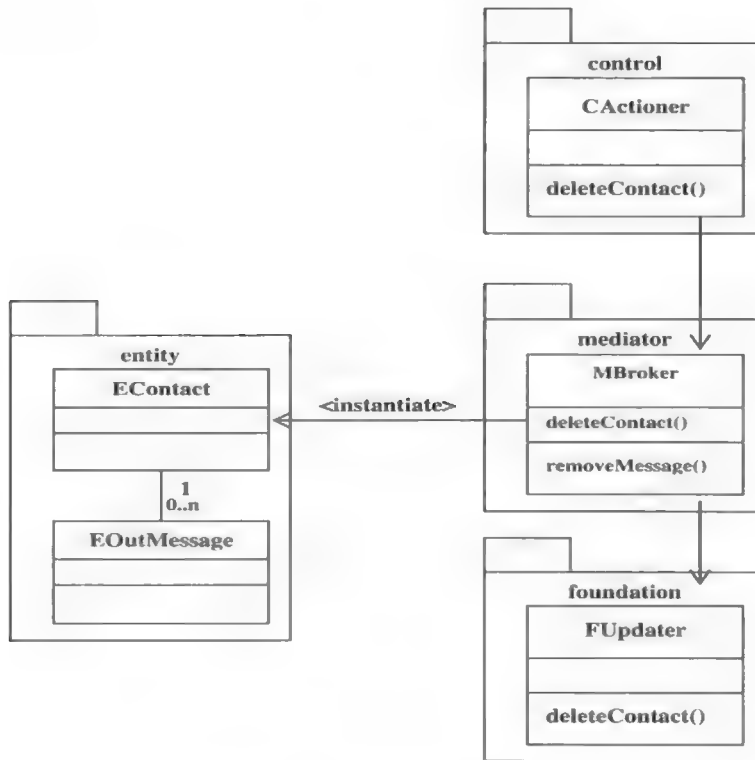
٥-٣-٦ نمط الوسيط Mediator

يعتبر الغرض من استخدام نمط الوسيط هو حل مشكلة الاقتران المباشر بين الكائنات الموجودة في طبقات مختلفة. يقدم هذا النمط مجموعة من الكائنات الوسيطة التي تقوم بعملية الاتصال بين الكائنات الموجودة في الطبقات المختلفة. لقد تم استخدام هذا النمط في إطار PCMEF، إذ تقوم طبقة Mediator بدور

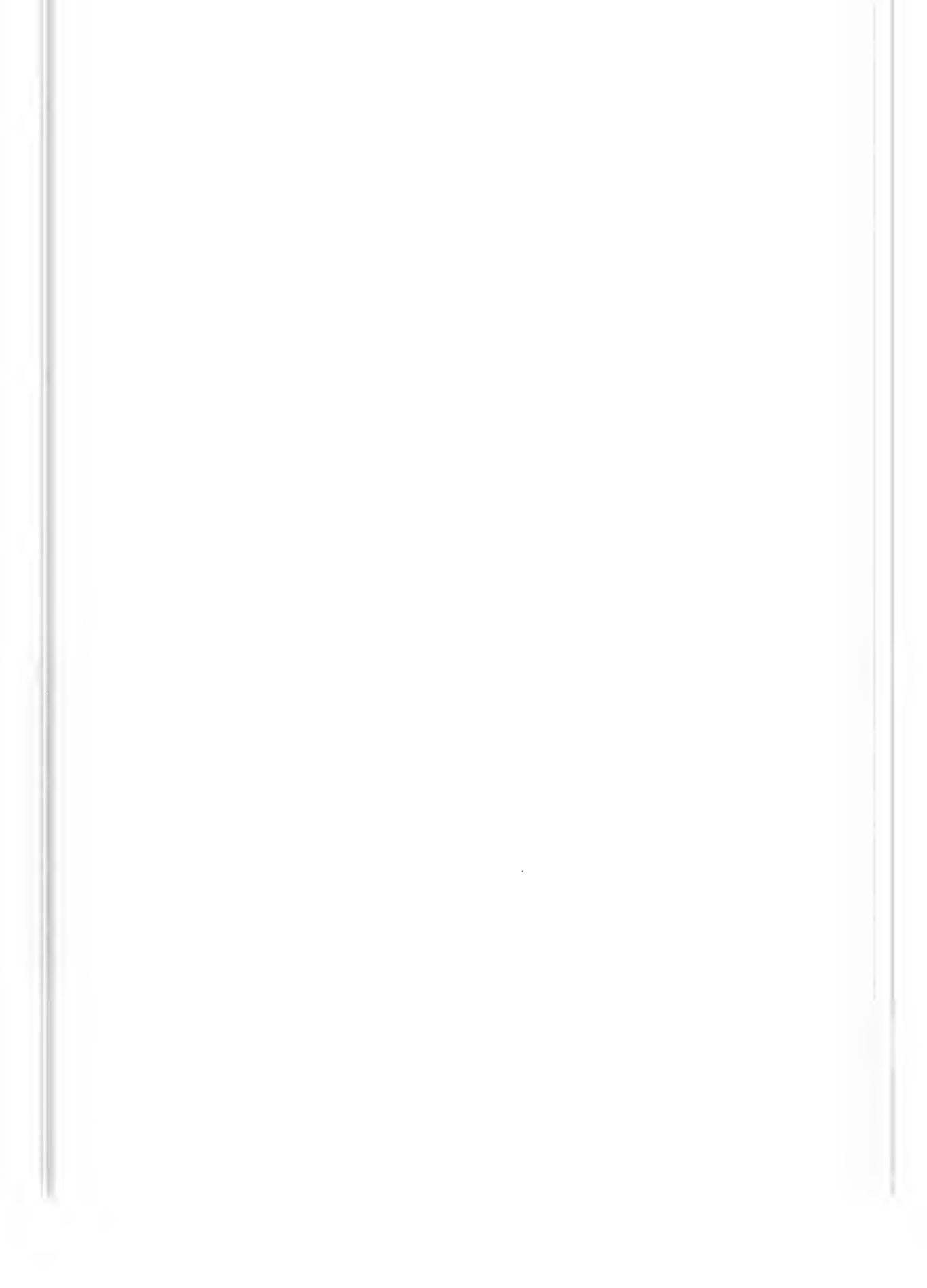
الوسيط بين طبقة الأساس Foundation وحزمة Entity. وتقوم طبقة التحكم Control أيضاً بدور الوسيط بين طبقة العرض Presentation وحزمة Entity. يقوم نمط الوسيط بدور آخر مهم وهو ضمان الحفاظ على التزامن بين كائن Subject وكائنات Observers عند حدوث تغييرات فيه. يوضح شكل (٦-١٧) مثالا لاستخدام نمط الوسيط، إذ يطلب الكائن CActioner من الكائن الوسيط MBroker القيام بحذف أحد الكائنات EContact. يقوم الكائن الوسيط بمهمتين هما:

١- يمرر رسالة حذف الكائن EContact إلى الكائن FUpdater الموجود في طبقة الأساس لتنفيذ عملية حذف الكائن من قاعدة البيانات.

٢- يقوم بالتحقق من وجود الكائن EContact في الذاكرة، فإذا كان موجوداً فإن الكائن الوسيط يقوم بإرسال رسالة لحذفه من الذاكرة أيضاً.



شكل (٦-١٧) نمط الوسيط



الفصل السابع

تصميم قواعد البيانات

Database Design

تعتبر البيانات Data هي الرافد الأساسى لنظم المعلومات، إذ يتم تخزينها فيما يسمى بقواعد البيانات Database ليتم التعامل معها من خلال برامج التطبيق. تقوم برامج التطبيق بالاتصال بقاعدة البيانات لاسترجاع البيانات المطلوب معالجتها وإجراء التعديلات عليها، ثم إعادة تخزينها مرة أخرى فى قاعدة البيانات. وأحيانا تقوم بعض برامج التطبيق بالاستعلام فقط عن بعض البيانات المخزنة بقواعد البيانات من دون إجراء تعديلات عليها. وعلى هذا فإنه يمكن للعديد من البرامج المختلفة أن تتعامل مع قاعدة بيانات معينة فى الوقت نفسه. أى أن قاعدة البيانات تقوم بدور الخادم Server للعديد من التطبيقات التى تقدم خدمات للعديد من المستخدمين من خلال محطات عمل مختلفة Client Workstations مرتبطة بقاعدة البيانات. هناك مجموعة من البرمجيات التى تمكننا من التعامل مع قواعد البيانات وتكون هى المسؤولة عن عمليات التخزين واسترجاع ومعالجة البيانات ويطلق عليها اسم «نظام إدارة قواعد البيانات (Database Management System (DBMS». وهناك أيضا لغة برمجة قياسية تستخدم للتعامل مع قواعد البيانات يطلق عليها اسم «لغة الاستفسار الهيكلية (Structured Query Language (SQL».

تتسم قواعد البيانات الجيدة بالخصائص التالية:

- كبيرة الحجم Large Size.
- دائمة Persistent.
- قابلة للاشتراك للعديد من المستخدمين Multi-User Shareable.
- قابلة للاسترداد Recoverable.
- متسقة Consistent.
- آمنة Secure.
- قابلة للاتساع Extensible.

٧-١ قواعد البيانات المرتبطة Relational Databases:

تستخدم قواعد البيانات المرتبطة على نطاق واسع منذ أكثر من عشرين عاماً لتخزين ومعالجة بيانات مشاريع نظم المعلومات. يقوم نموذج البيانات المرتبط Relational Data Model بتمثيل البيانات في شكل سجلات Records أو صفوف Rows داخل مجموعة من الجداول المرتبطة معاً في شكل علاقات منطقية تربط بين الجداول عن طريق بعض الخصائص المشتركة. يعتمد هذا النموذج بشكل أساسي على مجموعة من العلاقات الرياضية بين المجموعات وكذلك العلاقات المنطقية بينها. سوف نستعرض الآن أهم خصائص هذا النموذج وكيفية تطبيقه وبرمجته مع الاستعانة بحالة دراسية لتوضيح المفاهيم.

٧-١-١ الجدول Table:

يعتبر الجدول هو أحد المكونات الأساسية في نموذج قواعد البيانات المرتبطة. يتكون الجدول من مجموعة محددة من الأعمدة Columns وعدد غير محدد من الصفوف Rows. تمثل الأعمدة مجموعة الحقول أو الخصائص Attributes التي تميز عناصر الجدول. يوضح شكل (٧-١) هيكل أحد الجداول وهو يمثل بيانات الأفلام وذلك باستخدام نظام قواعد البيانات أوراكل ORACLE Database. يركز التصميم على تحديد مواصفات كل عمود في الجدول مثل: الاسم Column Name ونوع البيانات للعمود Data Type ونوع المفتاح المستخدم PK, FK وإمكانية أن يكون العمود خالياً من البيانات Null.

movie			
movie_code	Number(5)	<pk>	not null
movie_title	Varchar2(30)		null
director	Varchar2(20)		null

شكل (٧-١) تصميم جدول الأفلام

ويتطبيق قواعد المجموعات الرياضية فإن الجدول يحتوي على مجموعة مميزة من الصفوف أي أنه لا يمكن تكرار بيانات أحد الصفوف داخل الجدول. ولضمان ذلك يتم استخدام أسلوب المفتاح الأساسي (PK) Primary Key للجدول. يتكون المفتاح

الأساسى للجدول من عمود واحد أو أكثر وهى ذات قيم فريدة لا يمكن تكرارها أبداً داخل الجدول. ومن الطبيعى أن تكون هذه الأعمدة المكونة للمفتاح الأساسى غير خالية من البيانات NOT NULL. يمكننا تطبيق مواصفات التصميم للجدول باستخدام لغة الاستفسار الهيكلية SQL أو باستخدام إحدى أدوات نظم إدارة قواعد البيانات مثل ORACLE Schema Builder. يوضح شكل (٧-٢) جملة SQL المستخدمة لإنشاء الجدول حسب التصميم المبين بشكل (٧-١).

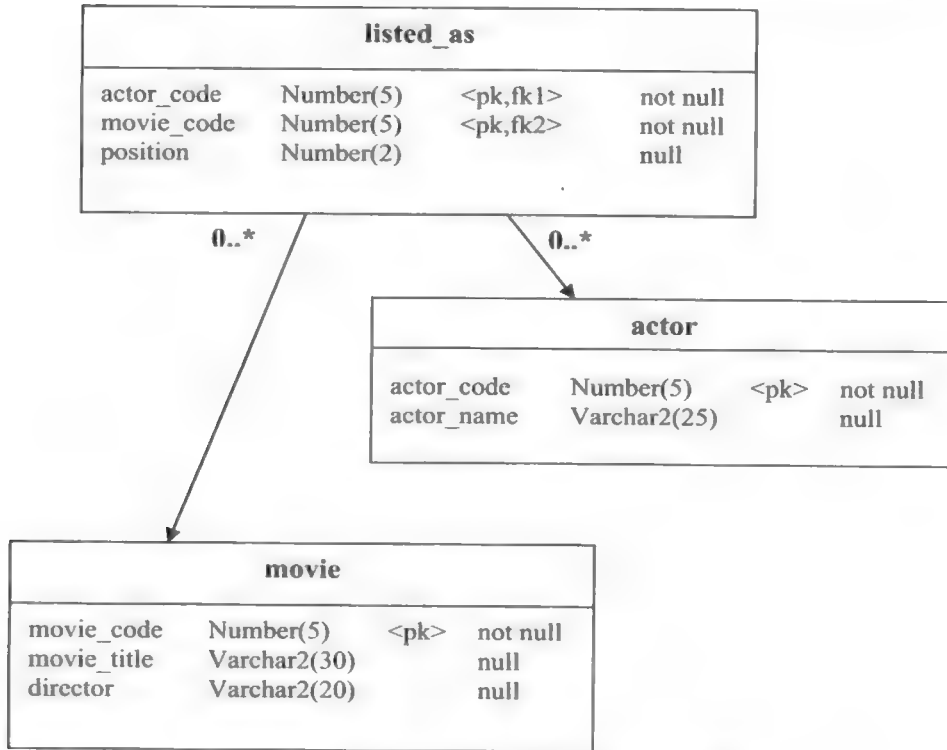
Create Table movie
<pre> Create Table Movie (Movie_Code Number(5) Not Null , Movie_Title Varchar2(30) , Director Varchar2(20) , Constraint PK_Movie Primary Key (Movie_Code)); </pre>

شكل (٧-٢) جملة SQL لإنشاء جدول الأفلام

٧-١-٢ التكامل المرجعى Referential Integrity

تتميز قواعد البيانات المرتبطة Relational Databases بتطبيقها لمفهوم الربط بين الجداول باستخدام أسلوب التكامل المرجعى. فمن المعروف أن عملية الربط بين الجداول تتم باستخدام نوعين من المفاتيح هما المفتاح الأساسى PK والمفتاح الخارجى FK، إذ يتم ربط أحد سجلات الجدول الذى يحتوى على المفتاح الأساسى بجميع السجلات المقابلة له فى الجدول المرتبط به، وتكون قيمة المفتاح الخارجى له تساوى قيمة المفتاح الأساسى. تكمن أهمية تطبيق مبدأ التكامل المرجعى إلى كونها تضمن سلامة بيانات الجداول المرتبطة، إذ إنه لا يمكن تسجيل بيانات فى الجدول إلا إذا كان يحتوى على قيمة للمفتاح الخارجى يوجد لها نظير مطابق للمفتاح الأساسى فى الجدول المرتبط به. يوضح شكل (٧-٣) نموذجاً منطقياً لقاعدة بيانات تتكون من ثلاثة جداول مرتبطة. يحتوى جدول Movie على بيانات الأفلام فقط ويحتوى جدول Actor على بيانات الممثلين، فى حين يقوم جدول Listed-As بدور مهم جداً

وهو تجميع البيانات المرتبطة بكلا الجدولين، حيث يحتوى على بيانات دور كل ممثل في الأفلام التي اشترك فيها. نلاحظ في المثال أن المفتاح الأساسي للجدول Listed-As يتكون من المفتاحين الخارجيين Fk1, Fk2 في شكل مفتاح أساسي مركب، إذ إنه لا يمكن تسجيل بيانات تمثل اشتراك الممثل نفسه في الفيلم نفسه مرة أخرى. يتضح من العلاقة أن الفيلم الواحد يحتوى على مجموعة من الممثلين، كما أن كل ممثل يمكن أن يشارك في أكثر من فيلم. ونلاحظ أيضاً أن قيم كل من المفاتيح الأساسية والمفاتيح الخارجية لا يمكن أن تكون خالية Not Null.

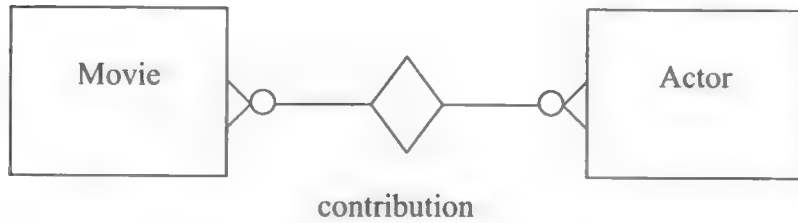


شكل (٧-٣) نموذج منطقي لقاعدة بيانات الأفلام

٧-٣-١ النموذج المفاهيمي Conceptual Model

تمر عملية نمذجة قواعد البيانات بعدة مستويات من التجريد Abstraction. يعتمد كل مستوى على درجة التفاصيل التي يجب أن يقدمها النموذج. تتسم نمذجة

بيانات النظام فى المراحل الأولية للتصميم باعتمادها الأسلوب المفاهيمى لبيانات كائنات النظام والعلاقات الطبيعية بينها دون الدخول فى التفاصيل الخاصة بسلامة البيانات وتكاملها أو حتى إمكانية تطبيقها باستخدام إحدى أدوات إدارة قواعد البيانات. يعتبر مخطط العلاقة بين الكائنات (Entity Relationship Diagram (ERD هو أحد أساليب نمذجة قواعد البيانات بشكل مفاهيمى. يوضح شكل (٧-٤) النموذج المفاهيمى المقابل للنموذج المنطقى الموضح بشكل (٧-٣). نلاحظ من الشكل أن النموذج قد أخفى بعض التفاصيل التى تخص عملية التكامل المرجعى بين الجداول وأنه يركز فقط على إظهار البيانات الأساسية لكل جدول وتوضيح أنواع العلاقات الطبيعية بين الجداول.



شكل (٧-٤) النموذج المفاهيمى لقاعدة بيانات الأفلام

٧-١-٤ النموذج المنطقى Logical Model

يهتم النموذج المنطقى بتقديم نموذج حقيقى قابل للتطبيق باستخدام إحدى أدوات إدارة قواعد البيانات. وعلى هذا فهو يهتم بما يلى:

- كائنات الربط Association Entity :

هى عبارة عن كائنات تمثل العلاقة بين كائنات أخرى وخصوصاً إذا كانت العلاقة تحتوى على خصائص تميزها. تستخدم هذه الكائنات أيضاً لتبسيط بعض العلاقات المركبة التى يصعب تطبيقها فعلياً باستخدام الأدوات المتاحة لإدارة قواعد البيانات مثل علاقة متعدد - إلى - متعدد أو علاقة بين ثلاثة كائنات أو أكثر.

- التكامل المرجعى Referential Integrity :

هو كما ذكرنا سابقاً يهتم بسلامة البيانات الموجودة فى الجداول المرتبطة.

- نوعية نظام إدارة قواعد البيانات DBMS:

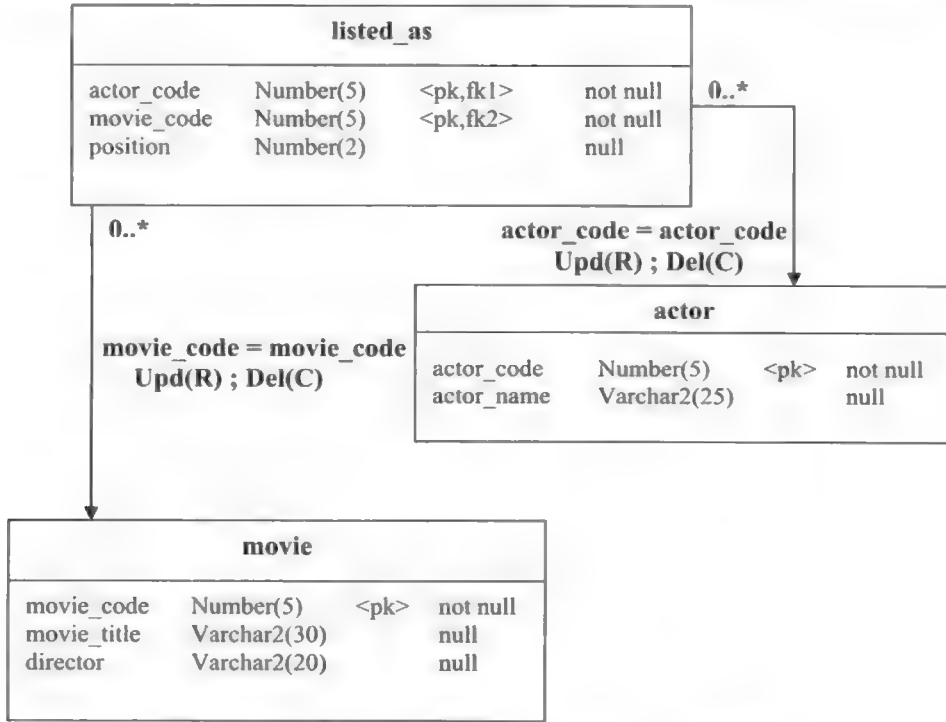
يتم تمثيل الجداول بالشكل الذى يلائم نوعية نظام إدارة قواعد البيانات المستخدم. يمثل النموذج الموضح بالشكل (٧-٣) أحد النماذج المنطقية لقاعدة بيانات الأفلام باستخدام نظام إدارة قواعد البيانات «أوراكل».

٧-١-٥ تطبيق قواعد العمل Implementing Business Rules

مع أنه قد تم الأخذ فى الاعتبار عند تصميم قواعد البيانات التأكد من سلامة بيانات الكائنات المرتبطة من خلال قيود التكامل المرجعى، إلا أنه يجب أيضاً التحقق من صحة البيانات من وجهة نظر نظام الأعمال وذلك من خلال تطبيق بعض قواعد العمل الضرورية الخاصة بطبيعة نظام الأعمال. أحيانا تكون بعض قواعد العمل بسيطة ويمكن تطبيقها على جميع السجلات بالجدول مثل إسناد قيمة افتراضية لأحد الحقول عند إنشاء سجل جديد (كأن يتم إسناد قيمة تاريخ اليوم إلى حقل التاريخ بالجدول). وكذلك إضافة قيد على عملية الحذف لأحد سجلات جدول مرتبط بآخر. فمثلا يمكننا إضافة قيد عمل خاص بقاعدة بيانات الأفلام يؤدي إلى حذف جميع السجلات الموجودة فى جدول Listed-As والخاصة بفيلم معين قد تم حذفه من جدول Movie.

يتم تمثيل معظم قواعد العمل البسيطة من خلال تعريف قواعد التكامل المرجعى فى أثناء عمليات الإضافة والتعديل والحذف. يوضح شكل (٧-٥) كيفية تمثيل بعض قواعد العمل البسيطة مثل Del (C) التى تعنى أنه فى حالة حذف سجل من جدول الأفلام يجب امتداد الحذف ليشمل السجلات المتعلقة به فى الجدول المرتبط به. أما Upd (R) فتعنى أن عملية التعديل تكون مقيدة، إذ لا يسمح نظام إدارة قواعد البيانات بتعديل كود فيلم معين فى جدول الأفلام وذلك للحفاظ على سلامة التكامل المرجعى بينها وبين الجداول المرتبطة به.

هناك أيضاً بعض قواعد العمل المركبة التى تتضمن تنفيذ بعض الإجراءات على البيانات المرتبطة بحدث معين. يتم تمثيل قواعد العمل المركبة فى شكل وحدات برمجية باستخدام إحدى لغات برمجة قواعد البيانات تسمى الزنادات «Triggers». يتم تطبيق هذه الوحدات البرمجية على الجدول المخصص له عند حدوث أى من العمليات الأساسية على قواعد البيانات مثل الإضافة والتعديل والحذف. يوضح شكل (٧-٦) مثالا لإحدى الوحدات البرمجية (الزناد) التى يتم تطبيقها على جدول الأفلام عند حذف فيلم معين.



شكل (٥-٧) قواعد العمل في النموذج المنطقي

Procedural referential integrity with delete trigger in MovieActor
<pre> Create trigger t_movie after delete On movie for each row Begin -- Delete all children in "Listed_As" Delete listed_as Where movie_code = :old.movie_code; End; </pre>

شكل (٦-٧) زناد الحذف على جدول الأفلام

٦-١-٧ برمجة تطبيقات قواعد البيانات Programming Database Applications

لا ينبغي النظر لقواعد البيانات باعتبارها مجموعة من الجداول الصماء ولكن في حقيقة الأمر أنها تكون نشطة ومتفاعلة مع التطبيقات المرتبطة بها من خلال مجموعة

مجموعة من البرامج. بعض هذه البرامج يكون مخزناً في قواعد البيانات نفسها والبعض الآخر يكون مخزناً في التطبيقات التي تستخدم قواعد البيانات. تتطلب برمجة تطبيقات قواعد البيانات استخدام إحدى لغات البرمجة التي تتضمن تعليمات لغة الاستفسار البنائية SQL. يتم إرسال كل جملة SQL إلى قواعد البيانات ليتم التحقق من صحتها وتفسيرها ومن ثم تنفيذها واسترجاع النتائج إلى التطبيق. إذا كان هناك العديد من جمل SQL التي تتطلب التعامل مع قواعد البيانات بشكل مستمر فسوف يؤثر ذلك في أداء النظام بشكل سلبي. وعلى ذلك فإنه يجب عدم التوسع في استخدام هذا الأسلوب إلا في الحالات التي تتطلب أن تتم برمجة قواعد البيانات من خلال واجهة التطبيق. يوضح شكل (٧-٧) مثلاً لهذا الأسلوب.

استخدام تعليمات SQL للبحث في قواعد البيانات من داخل تطبيق الأفلام

```
استعراض اسم الممثل صاحب دور البطولة لكل فيلم --
Execute Select movie_title , actor_name
      From movie m , listed_as l , actor a
      Where m.movie_code = l.movie_code
      AND l.actor_code = a.actor_code
      AND l.position = 1 ;
```

شكل (٧-٧) برمجة قواعد البيانات من خلال التطبيق

يتم في أغلب الأحوال برمجة تطبيقات قواعد البيانات من خلال مجموعة من الإجراءات المخزنة Stored Procedures في خادم قواعد البيانات نفسها ويتم استدعاؤها من خلال واجهات التطبيق. يوضح شكل (٧-٨) مثلاً لأحد الإجراءات المخزنة باستخدام لغة أوراكل وذلك للبحث في قاعدة بيانات الأفلام. نلاحظ من الشكل أن تنفيذ الإجراء يتم من خلال بيئة برنامج SQL ولكن في الحقيقة يكون جزءاً من برامج التطبيق المخزن عند العميل. ونلاحظ أيضاً أن الإجراء يحتوى على معامل إدخال Input Parameter يستخدم لتمرير متغيرات من التطبيق إلى الإجراء في أثناء التنفيذ. ويتم أيضاً استرجاع نتائج تنفيذ الإجراء إلى التطبيق المستدعى له.

استخدام أسلوب الإجراءات المخزنة للبحث في قواعد بيانات الأفلام

--

```
Create or Replace Procedure string_search(string IN varchar2) AS
Cursor c1 IS
    Select movie_title AS found
    From movie
    Where upper(movie_title) Like '%' || upper(string) || '%';
Begin
    For c1rec IN c1 Loop
        Dbms_output.put_line
            ('Found movie title: ' || LPAD(c1rec.found,30));
    End Loop;
End string_search;
```

```
SQL> Execute string_search( 'الخوف' );
Found movie title :    شيء من الخوف
```

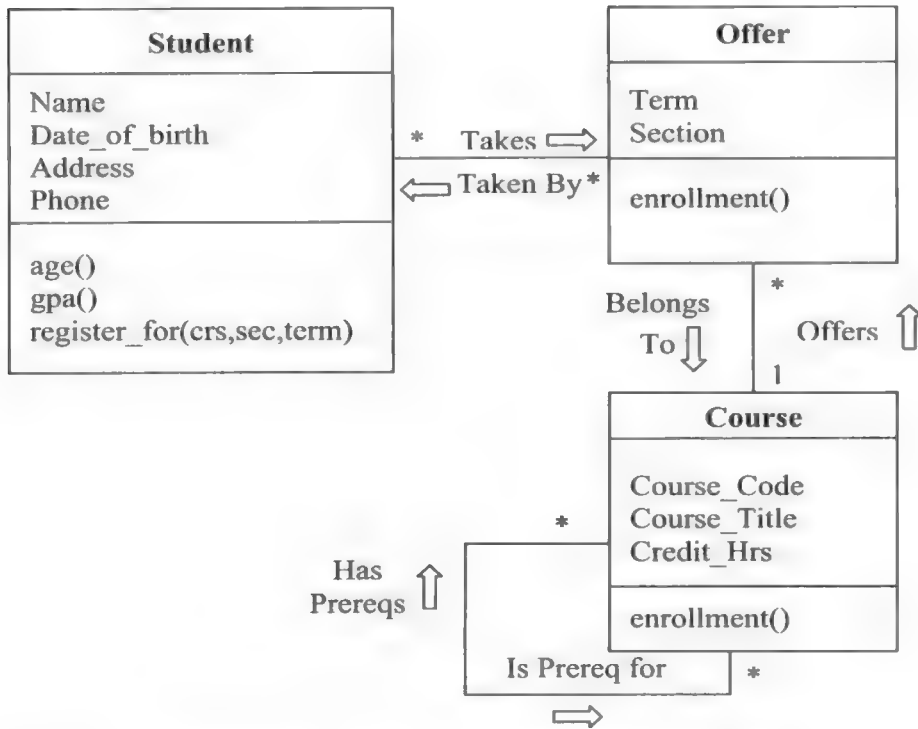
شكل (٧-٨) استخدام الإجراءات المخزنة Stored Procedures

٧-٢ قواعد البيانات الشيئية Object Databases:

مع زيادة استخدام أسلوب النمذجة الشيئية المعتمدة على الكائنات أصبح من الضروري أن يتم اعتماد أسلوب لبناء قواعد بيانات معتمدة على الكائنات. وقد قامت مجموعة ODMG في منتصف التسعينيات باعتماد معايير قياسية لقواعد البيانات المعتمدة على الكائنات OODB. وقد استحدثت المجموعة في الفترة الأخيرة معايير أخرى تقوم على التحول من توصيف نظام إدارة قواعد البيانات المعتمدة على الكائنات إلى تعريف طبقة برمجية جديدة تسمى «Object Storage API» بين برامج التطبيق وقواعد البيانات المتصلة بها. تقوم هذه الطبقة بدور التناظر بين كائنات التطبيق وبين السجلات في جداول قواعد البيانات. يتم اعتماد هذا الأسلوب ليلائم الإجراءات الروتينية المستخدمة في إدارة قواعد البيانات.

تقدم مجموعة ODMG لغة استفسار معتمدة على الكائنات تسمى (OQL) Object Query Language مشابهة للغة الاستفسار البنائية SQL المستخدمة في قواعد البيانات المرتبطة. يمكننا أيضاً استخدام لغة تعريف الكائنات (ODL) Object Definition Language لتوصيف هيكل قواعد البيانات المعتمدة على الكائنات (Hoffer et al, 2002).

ولتوضيح كيفية استخدام هذه اللغة سوف نستعين بمثال لنمذجة الأصناف في تطبيق التحاق الطلاب بالجامعة باستخدام لغة النمذجة الموحدة UML كما هو موضح بالشكل (٧-٩). يمثل هذا المخطط النموذج المفاهيمي لقواعد بيانات النظام.



شكل (٧-٩) النموذج المفاهيمي لقواعد بيانات الجامعة

تستخدم لغة ODL بعض الكلمات المحجوزة لتعريف كل من خصائص Attributes الكائنات والعلاقات بينها. وتستخدم أيضاً الشكل العام للدوال في تعريف الطرق Methods. يوضح شكل (٧-١٠) استخدام لغة ODL في توصيف قواعد بيانات الجامعة المقابل للنموذج المفاهيمي الموضح في شكل (٧-٩).

```

Class Student {
    ( extent students )
    Attribute string name;
    Attribute date date_of_birth;
    Attribute Address address;
    Attribute Phone phone;
    Relationship set (Offer) takes inverse Offer::taken_by;
    Short age();
    Float gpa();
    Boolean register_for(string crs, short sec, string term);
};

Class Offer {
    ( extent offers )
    Attribute string term;
    Attribute enum section {1,2,3,4,5,6,7,8};
    Relationship set (Student) taken_by inverse Student::takes;
    Relationship Course belongs_to inverse Course::offers
    Short enrollment();
};

Class Course {
    ( extent courses )
    Attribute string course_code;
    Attribute string course_title;
    Attribute short credit_hrs;
    Relationship set (Course) has_prereqs inverse Course::is_prereq_for;
    Relationship set (Course) is_prereq_for inverse Course::has_prereqs;
    Relationship list (Offer) offers inverse Offer::belongs_to;
    Short enrollment();
};
    
```

شكل (٧-١٠) استخدام لغة ODL لتوصيف قاعدة بيانات الجامعة

٧-٣ قواعد البيانات الشيئية-المرتبطة Object-Relational Databases:

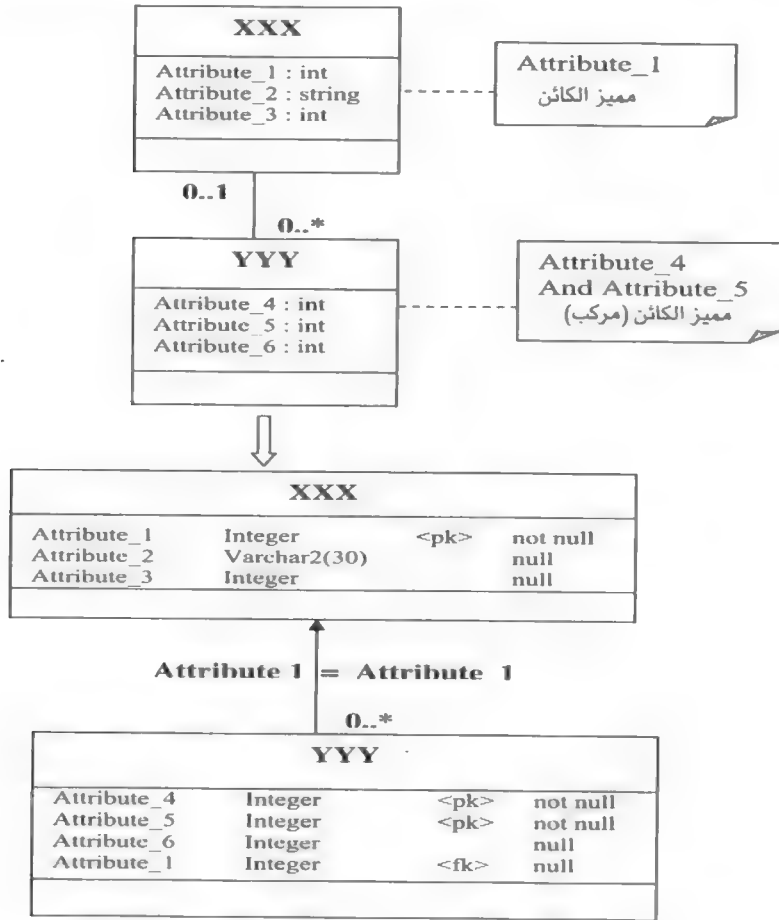
تحتوى كل من قواعد البيانات المرتبطة RDB وقواعد البيانات الشيئية ODB على مجموعة من المميزات ونقاط الضعف. ويلاحظ أن نقاط الضعف فى أى منها تمثل

مميزات النوع الآخر. وبالطبع فقد أدى ذلك إلى ظهور نوع جديد من قواعد البيانات يسمى (Object-Relational (ORDB وهو يحتوى على مميزات كل من النوعين السابقين. تعتبر النسخة ORACLE 9i لقواعد البيانات أوراكل هى أحد أنواع نظم إدارة قواعد البيانات الشبيهة-المرتبطة ORDBMS. يدعم هذا النوع من قواعد البيانات عمليات تعريف كل من البيانات المرتبطة Relational Data والكائنات Objects، بالإضافة إلى إمكانية التعامل معها والاستفسار عنها وذلك من خلال لغة موحدة وواجهة مشتركة. يأخذ النموذج الأساسى للبيانات الشكل العلقى (الارتباطى Relational)، فى حين تأخذ العمليات والإجراءات الشكل الشيئى (المبنى على الكائنات Object-Oriented). تتم عمليات تخزين واسترجاع البيانات باستخدام قاعدة بيانات مرتبطة. ويتطلب ذلك إجراء عملية تحويل تناظرى بين الأسلوبين (Object, Relational). وقد استلزم ذلك إصدار نسخة مطورة من لغة الاستفسار لتلائم طبيعة التعامل مع كل من نوعى البيانات. يدعم هذا الأسلوب استخدام المفاهيم الأساسية للأسلوب الشيئى مثل: التوريث والتجريد واستخدام أنواع معقدة من البيانات مثل ملفات الصوت والصورة وغيرها، بالإضافة إلى دعمه لأسلوب التكامل المرجعى للتأكد من سلامة البيانات المخزنة فى الجداول المرتبطة. وجدير بالذكر أن عملية التحويل بالتناظر بين الكائنات والجداول المرتبطة يمكن أن تتم فى الاتجاهين. فمن الممكن أن يتم التحويل من الكائنات إلى الجداول المرتبطة وذلك فى عند إنشاء قاعدة البيانات بناءً على نموذج الأصناف للتطبيق. ويمكن أن يتم التحويل من الجداول المرتبطة إلى الكائنات وذلك فى أثناء معالجة البيانات المستخرجة من قاعدة البيانات إلى ذاكرة التطبيق. غالباً ما تتم عملية تحويل الكائنات إلى الجداول المناظرة لها بسهولة أكبر من عملية تحويل العلاقات بين الكائنات إلى العلاقات المناظرة لها بين الجداول. سوف نستعرض الآن عملية تحويل العلاقات الأكثر استخداماً فى نمذجة قواعد البيانات.

تحويل علاقة واحد - إلى - متعدد:

تعتبر علاقة واحد - إلى - متعدد من أكثر العلاقات شيوعاً ومن أكثرها سهولة أيضاً فى أثناء عملية التحويل إلى العلاقة المقابلة بين الجداول. يتم تحويل الكائن ناحية طرف العلاقة الأحادية إلى جدول يحتوى على مجموعة من الأعمدة المقابلة لمجموعة خصائص الكائن Attributes مع تحديد المفتاح الأساسى الذى يميز جدول

الكائن. ويتم تحويل الكائن ناحية طرف العلاقة المتعددة إلى جدول آخر يضم مجموعة خصائص الكائن مع إضافة عمود يمثل المفتاح الخارجي للجدول FK يرتبط بالمفتاح الأساسي في الجدول السابق. يوضح شكل (٧-١١) مثلاً على هذا التحويل.

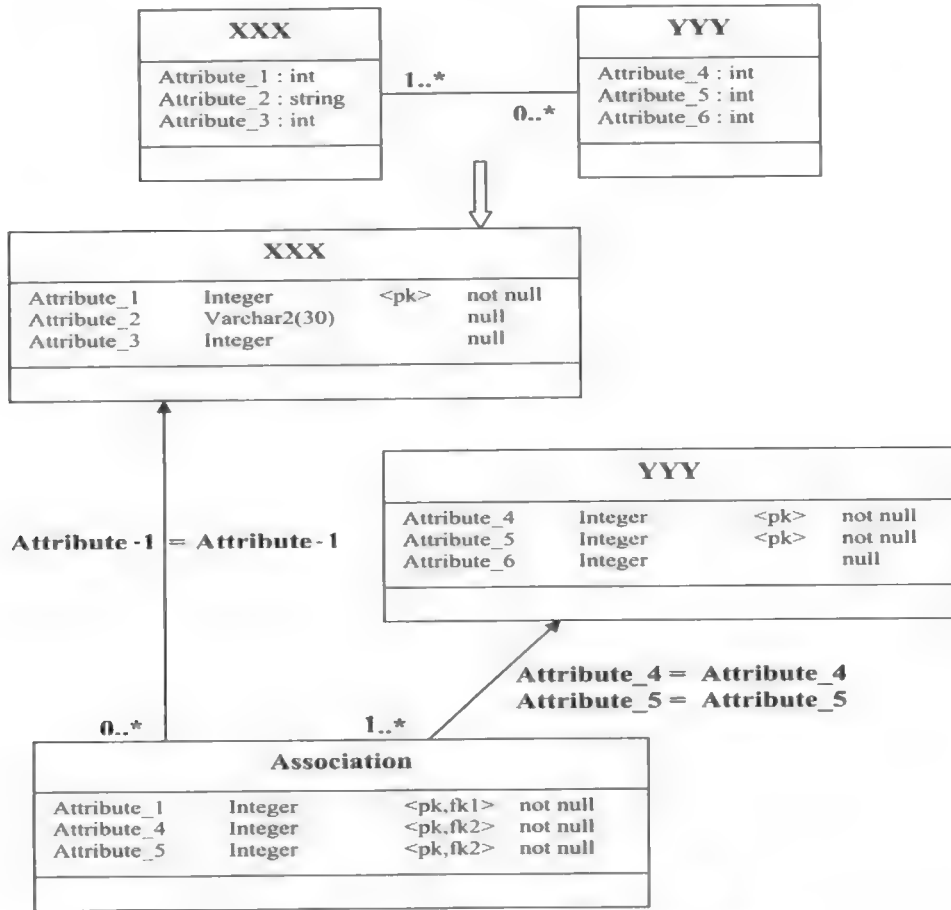


شكل (٧-١١) تحويل علاقة واحد - إلى - متعدد

تحويل علاقة متعدد - إلى - متعدد

من المعروف أن قواعد البيانات المرتبطة لا تستطيع تطبيق العلاقة متعدد - إلى - متعدد مباشرة، ولكن يتم إضافة جدول خاص لتمثيل تلك العلاقة وتحويلها إلى

علاقته من النوع واحد - إلى - متعدد. يحتوى جدول العلاقة على مجموعة من الأعمدة تمثل المفاتيح الخارجية المقابلة للمفاتيح الأساسية فى الجدولين الأساسيين كما هو موضح بالشكل (١٢-٧).

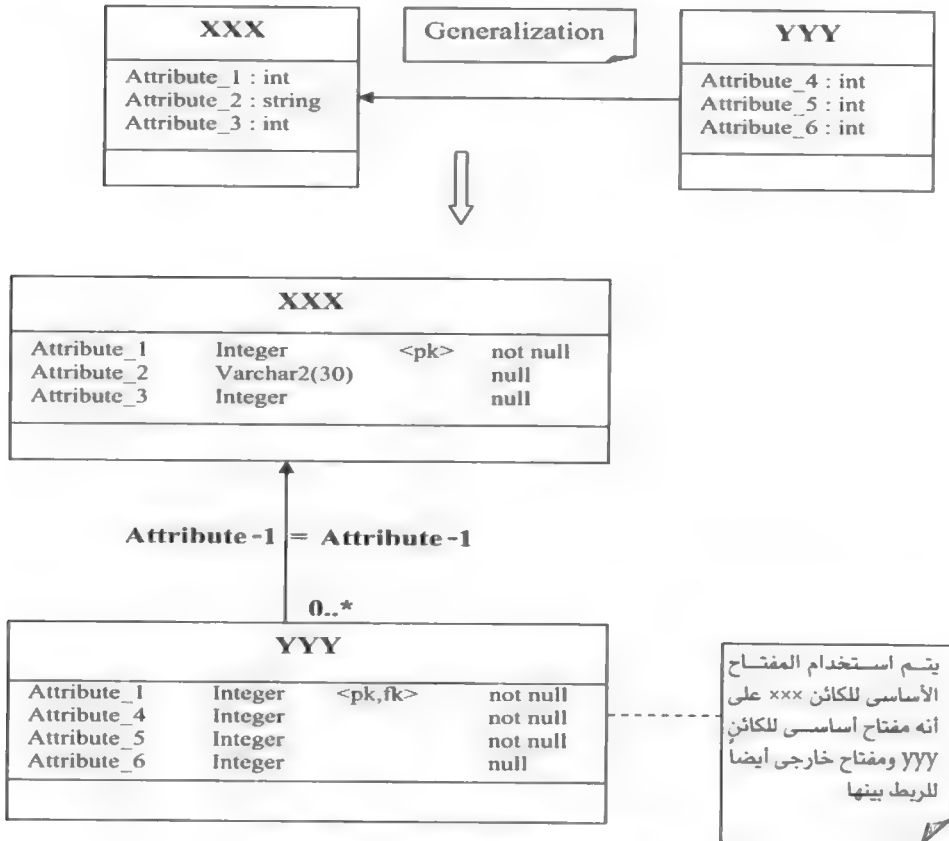


شكل (١٢-٧) تحويل العلاقة متعدد - إلى - متعدد

تحويل علاقة التعميم:

لا يستطيع النموذج العلاقى أن يدعم مفهوم التوريث الخاص بالنموذج الشيئى. ولحل هذه المشكلة يمكننا أن نكرر جميع الخصائص الموجودة فى الصنف العام

Superclass وتضمينها في الجدول الذي يمثل الصنف الفرعى Subclass ونكتفى بهذا الجدول الذي يمثل كلا من الصنف العام والصنف الفرعى. هناك حل آخر موضح بالشكل (١٣-٧) وهو يقوم على تحويل الصنف العام إلى جدول مماثل وتحويل الصنف الفرعى إلى جدول آخر مع إضافة عمود يمثل المفتاح الأساسى للجدول وهو المفتاح الأساسى نفسه بالجدول الأول.

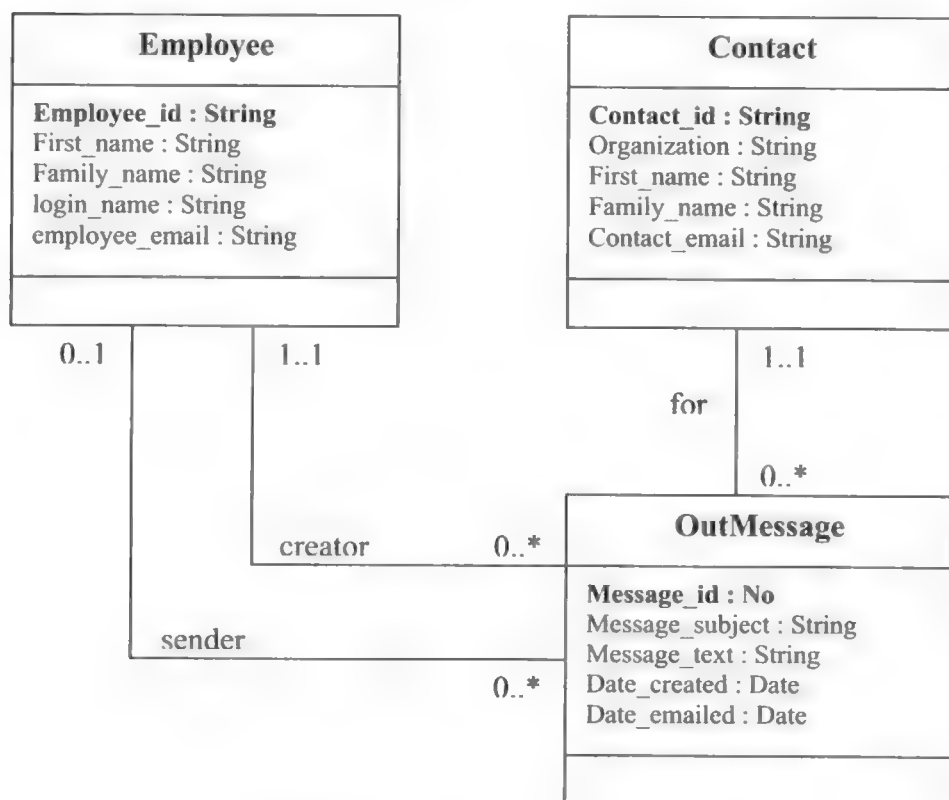


شكل (١٣-٧) تحويل علاقة التعميم

٤-٧ تصميم قواعد البيانات Database Design:

سنقوم الآن بتصميم قواعد البيانات الخاصة بالحالة الدراسية (إدارة البريد الإلكتروني EM) التى بدأناها فى الفصل الخامس من هذا الكتاب. تقوم برامج

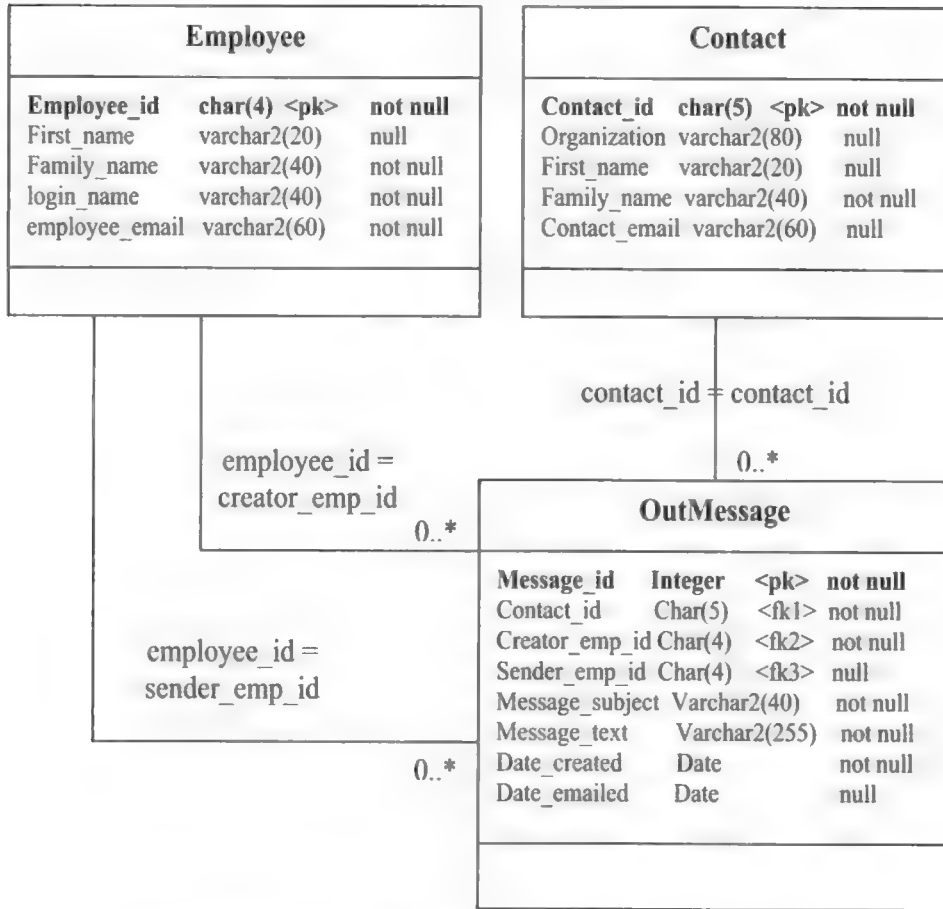
الحالة الدراسية بعمليات استخراج البيانات من قواعد البيانات وتجهيزها ثم إرسال الرسائل للعملاء، وبعد ذلك يتم تحديث قواعد البيانات بتمييز الرسائل التي تم إرسالها فعلاً وذلك بإضافة معلومات عن عملية الإرسال مثل الوقت. يوضح شكل (١٤-٧) مخطط الأصناف للتطبيق EM.



شكل (١٤-٧) مخطط الأصناف للتطبيق EM

١-٤-٧ نموذج قواعد البيانات Database Model،

يتم استنتاج نموذج قواعد البيانات من نموذج الأصناف للتطبيق بطريقة مباشرة وسهلة كما هو مبين بشكل (١٥-٧). يمثل هذا المخطط النموذج المادي لقواعد بيانات التطبيق Physical Database Model، حيث إنه يحدد الخصائص ونوعها باستخدام نظام معين لإدارة قواعد البيانات وهو نظام «أوراكل».



شكل (٧-١٥) النموذج المادي لقواعد بيانات التطبيق EM

٧-٤-٢ إنشاء مخطط قواعد البيانات Creating Database Schema:

الخطوة التالية لتصميم قواعد البيانات تتضمن مجموعة تعليمات لغة SQL الخاصة بإنشاء الجداول والعلاقات بينها بناء على النموذج المادي لقواعد البيانات. يمكننا استخدام إحدى أدوات هندسة البرمجيات Case Tools لتوليد الملف الخاص بتعليمات إنشاء قواعد البيانات من نموذج قواعد البيانات. يتم تنفيذ هذا الملف لإنشاء هيكل الجداول المرتبطة بقاعدة البيانات للتطبيق. يوضح شكل (٧-١٦) محتوى ملف تعليمات SQL الخاصة بإنشاء مخطط قواعد البيانات للتطبيق EM.

ملف يحتوى على تعليمات SQL لإنشاء قواعد بيانات التطبيق EM

```
-- Table : Contact
Create table Contact (
    Contact_id    char(5)          not null,
    Organization  varchar2(80),
    First_name    varchar2(20),
    Family_name   varchar2(40)     not null,
    Contact_email varchar2(60),
    Constraint PK_CONTACT primary key (contact_id) )
/

-- Table : Employee
Create table Employee (
    Employee_id   char(4)          not null,
    First_name    varchar2(20),
    Family_name   varchar2(40)     not null,
    Login_name    varchar2(40)     not null,
    employee_email varchar2(60)     not null,
    Constraint PK_EMPLOYEE primary key (employee_id) )
/

-- Index : Login_UN
Create unique index login_UN on Employee (
    Login_name ASC )
/

-- Table : Outmessage
Create table outmessage (
    Message_id    integer          not null,
    Contact_id     char(5)          not null,
    Creator_emp_id char(4)          not null,
    Sender_emp_id  char(4),
    Message_subject varchar2(40) not null,
    Message_text   varchar2(255) not null,
    Date_created   date             not null,
    Date_emailed   date,
    Constraint PK_OUTMESSAGE primary key (message_id),
    Constraint FK_OUTMES_REF_CONTACT foreign key (contact_id)
    references Contact (contact_id),
    Constraint FK_OUTMES_REF_CREATOREMP foreign key
    (creator_emp_id) references Employee (employee_id),
    Constraint FK_OUTMES_REF_SENDEREMP foreign key (sender_emp_id)
    references Employee (employee_id) )
/
```

شكل (١٦-٧) ملف تعليمات SQL لإنشاء قواعد بيانات التطبيق EM

٧-٤-٣ محتوى عينة قواعد البيانات Sample Database Content

بعد إنجاز عملية إنشاء مخطط قواعد البيانات بنجاح، سنبدأ بتحميل قواعد البيانات بالبيانات الفعلية وذلك في أثناء تنفيذ التطبيق. يمكننا أيضاً تحميل الجداول بالبيانات عن طريق تعليمات الإضافة الخاصة بلغة SQL. يوضح شكل (٧-١٧) بعض تعليمات SQL التي تقوم بالاستعلام واسترجاع البيانات الخاصة بالتطبيق EM.

SQL> select * from E,ployee;

Employee_id	First_name	Family_name	Login_name	Employee_email
1001	Ahmed	Al_Ghamdy	GhamdyA	Ghamdy@ipa.edu.sa
1002	Nasser	Al_Meshary	MesharyN	Meshary@ipa.edu.sa
1003	Alaa	Al_Sammak	SammakA	Sammak@ipa.edu.sa

SQL> select * from Contact;

Contact_id	Organization	First_name	Family_name	Contact_email
1234	STC	Ahmed	Hesham	HeshamA@stc.net.sa
1235	KSU	Saleh	Al_Zahrany	Zahrany@ksu.edu.sa
1236	KFU	Kamal	Al_Dahshan	Dahshan@kfu.edu.sa

SQL> select contact_id, creator_emp_id, sender_emp_id, messaga_subject, message_text from Outmessage order by contact_id;

Contact_id	Creator_emp_id	Sender_emp_id	Message_subject
------------	----------------	---------------	-----------------

Message_text

1234	1001	استعلام عن خدمات جديدة: الرجاء التكرم بموافقاتنا بمعلومات مفصلة عن الخدمات الجديدة التي تقدمها الشركة للعملاء، مع توضيح المزايا الخاصة بكل خدمة وأسعار تقديمها.
1235	1002	مطلب تقديم بحث: مرفق مع الرسالة نسخة من البحث المقدم للنشر في مجلة البحوث العلمية الخاصة بالجامعة، يرجى إحاطتنا بنتيجة فحص البحث فور الانتهاء من تقييمه.
1236	1003	استعلام عن برنامج: حيث إنتى أنهيت دراسة البكالوريوس. تخصص علوم الحاسب، بدرجة جيد جداً، وأود الالتحاق ببرنامج الدراسات العليا لديكم لدراسة الماجستير في التخصص نفسه، أود التكرم بتزويدنا بتفاصيل البرنامج ومواعيد الالتحاق ورسوم الدراسة.

شكل (٧-١٧) محتوى عينة قاعدة البيانات EM

الفصل الثامن

تصميم الأصناف والتفاعلات

Classes and Interactions Design

تركز أنشطة تصميم الأصناف على عمليات التأكد من تحقيق الأصناف للمتطلبات الوظيفية المحددة في نموذج حالات الاستخدام، بالإضافة إلى ملاءمتها للتصميم المعماري للنظام. تبدأ عمليات تصميم الأصناف بتحديد الأصناف والأماكن المناسبة لها داخل الحزم Packages. وبعد ذلك يتم تحديد خصائص وعمليات الأصناف التي تحقق المواصفات المطلوبة في نموذج حالات الاستخدام للتطبيق. وبعد ذلك يتم تأسيس العلاقات والتبعيات بين الأصناف. في الواقع لا يمكن فصل عملية تصميم التفاعلات عن تصميم الأصناف، فبمجرد انتهاء عملية تحديد الأصناف يمكننا البدء فوراً في تصميم التفاعلات التي سوف تساعدنا في تحديد مواصفات الأصناف وخصوصاً العمليات.

٨-١ إيجاد الأصناف من حالات الاستخدام Finding Classes from Use Cases:

تتطلب عملية تصميم الأصناف أن تحقق مواصفات المتطلبات المذكورة في نموذج حالات الاستخدام. ولذلك فإنه من الطبيعي أن يتم استخراج تلك الأصناف من الوثائق المرفقة بنماذج حالات الاستخدام التي تحتوي على المواصفات التفصيلية للمتطلبات الوظيفية لكل حالة استخدام. هناك أسلوب يعتمد على قائمة المتطلبات الموجودة في حالات الاستخدام ويستخدمها في استخراج الأصناف وذلك في شكل جدول يتكون من خمسة أعمدة كما يلي:

- ١- رقم المتطلب Requirement Number.
- ٢- تعريف المتطلب Requirement Definition.
- ٣- اسم الصنف والحزمة التابع لها Responsible Package and Class Name.
- ٤- اسم العملية للمتطلب Operation Name.
- ٥- اسم الحزمة والصنف المتعاون Collaboration Package and Class.

يوضح شكل (٨-١) جزءاً من الجدول المستخدم لإيجاد الأصناف من متطلبات حالات الاستخدام للتطبيق الخاص بإدارة البريد الإلكتروني EM.

رقم المتطلب	تعريف المتطلب	الحزمة / الصنف	اسم العملية	الحزمة/الصنف المتعاون
R1	يعرض النظام رسالة توضيحية للموظف لإدخال اسم المستخدم username وكلمة المرور password	Presentation PConsole Control CActioner	displayLogin getUserInput login	Presentation PConsole Control CActioner Foundation FConnection
R2	يقوم النظام بتنفيذ عملية اتصال الموظف بقاعدة بيانات النظام	Foundation FConnection Foundation FReader	getConnection readEmployee	Acquaintance IAconstants Foundation FReader
R3	يقوم النظام بعرض قائمة الاختيارات المتاحة للموظف	Presentation PConsole	displayMenu	
R4	عندما يختار الموظف الاختيار الرابع بالقائمة يتم الخروج من النظام	Presentation PConsole Control CActioner	getUserInput exit	Control CActioner Foundation FConnection
R5	عندما يختار الموظف الاختيار الأول يتم عرض رسالة بعدد الرسائل غير المرسلة، وعرض معلومات عن كل رسالة، ثم عرض قائمة الاختيارات مرة أخرى	Presentation PConsole Mediator MBroker Foundation FReader	getUserInput viewMessages retrieveMessages readMessages readContact	Presentation PConsole Acquaintance IAconstants Mediator MBroker Foundation FReader

Presentation PConsole Mediator MBroker	getUserInput displayMessageText	Presentation PConsole	عندما يختار الموظف الاختيار الثاني، يقوم الموظف بإدخال رقم الرسالة، ثم يقوم النظام بعرض محتوى الرسالة، ثم عرض قائمة الاختيارات مرة أخرى	R6
Presentation PConsole Control CActioner Acquaintance IAconstants Foundation FWriter	getUserInput prepareMessage sendMessage updateMessage displayConfirmation	Presentation PConsole Control CActioner Foundation FWriter Presentation PConsole	عندما يختار الموظف الاختيار الثالث، يقوم الموظف بإدخال رقم الرسالة، ثم يقوم النظام بإرسال الرسالة وتعديل قاعدة البيانات وعرض رسالة تأكيد بإرسال الرسالة، ثم عرض قائمة الاختيارات مرة أخرى	R7
Presentation PConsole Ref. R1 Control CActioner Foundation FConnection	displayLoginError exit	Ref. R1 Presentation PConsole Control CActioner	في حالة إدخال الموظف بيانات غير صحيحة لاسم المستخدم أو كلمة المرور، يقوم النظام بعرض رسالة خطأ، ويسمح للموظف بإعادة الإدخال مرة أخرى (عدد المرات المسموح بها = ٢)	R8
Presentation PConsole Foundation FConnection	getUserInput displayMenu logout	Presentation PConsole Control CActioner Ref. R1	في حالة إدخال الموظف رقم اختيار غير صحيح، يقوم النظام بتجاهل الاختيار ويعيد عرض قائمة الاختيارات مرة أخرى أو الخروج بعد ٢ محاولات خطأ	R9

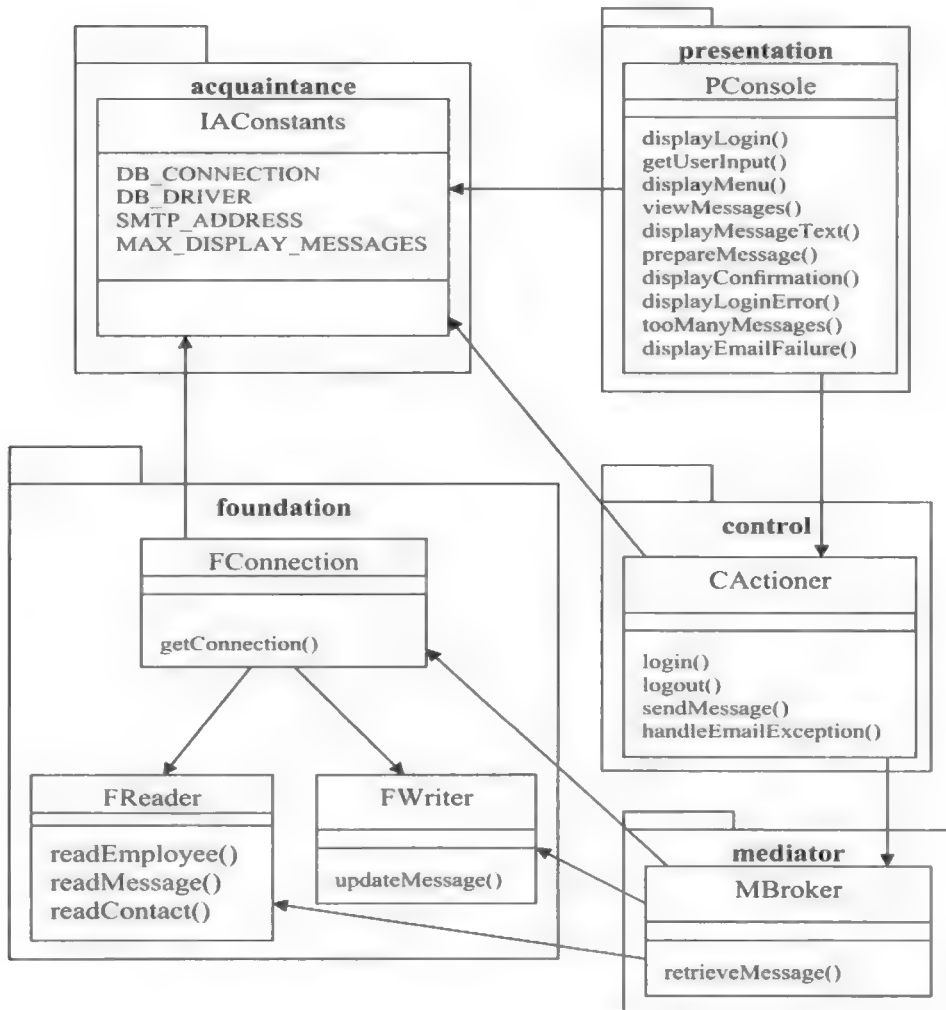
	tooManyMessages	Ref. R5 Presentation PConsole	في حالة اختيار الموظف الاختيار الأول ويكون عدد الرسائل غير المرسله أكبر من العدد المسموح بعرضها، يقوم النظام بعرض عدد الرسائل المسموح بها، ثم عرض رسالة توضيحية تفيد بوجود عدد آخر من الرسائل غير المرسله موجودة بقاعدة البيانات	R10
Presentation PConsole	andleEmailException displayEmailFailure displayMenu	Ref. R7 Control CActioner Presentation PConsole	في حالة فشل عملية إرسال الرسالة (الاختيار الثالث) يقوم النظام بعرض رسالة توضيحية للموظف تفيد بعدم نجاح عملية إرسال الرسالة، ثم يقوم بعرض قائمة الاختيارات مرة أخرى	R11

شكل (٨-١) جدول إيجاد الأصناف من حالات الاستخدام للتطبيق EM

٨-٢ تصميم الأصناف Classes Design:

يحتوى الجدول السابق على معلومات كافية لإعداد التصميم الأولي لنموذج الأصناف للتطبيق EM. يمثل النموذج الموضح بشكل (٨-٢) النموذج الأولي للأصناف والمقابل لجدول إيجاد الأصناف الموضح بشكل (٨-١). يعتبر هذا النموذج أولي لأنه لا يحتوى على جميع العلاقات بين الأصناف نتيجة عدم توافر جميع أنواع الرسائل المتبادلة بين الكائنات فى جدول إيجاد الأصناف. يتضمن النموذج الأولي الأصناف المذكورة فى جدول إيجاد الأصناف ومكانها داخل الحزم وكذلك علاقات الارتباط بينها. نلاحظ من النموذج أن بعض الحزم تحتوى على صنف واحد فقط لأن النموذج لا يتضمن الأصناف الخاصة بالكائنات الخارجية. يحتوى النموذج أيضا على حزمة Acquaintance التى تحتوى على معظم الثوابت المستخدمة فى النظام مثل ثوابت الاتصال بقواعد البيانات Database_Connection التى يتم استخدامها من خلال صنف الاتصال FConnection والثوابت الخاصة بخادم البريد SMTP_Address التى يتم استخدامها من خلال صنف التحكم CActioner الخاص بالتطبيق، والثابت الخاص بأقصى عدد ممكن للرسائل التى يمكن عرضها على الشاشة Max_Display_Messages التى يتم استخدامها من خلال صنف الواجهة PConsole.

يحتاج هذا النموذج الأولي إلى بعض التحسينات والإضافات لتتوافق مع المتطلبات غير الوظيفية الموضحة في إطار التصميم المعماري للنظام. يؤدي ذلك إلى توسعة النموذج الأولي استناداً إلى التصميم المعماري وذلك بإضافة أصناف جديدة وتعديلات في الأصناف المستندة إلى متطلبات المستخدم. تبدأ هذه العملية بجزء من جدول إيجاد الأصناف، وبعد ذلك يتم توسعته ليضم أعمدة أخرى تقدم الأصناف الجديدة والتعديلات على الأصناف الموجودة، كما هو موضح بشكل (٨-٣).



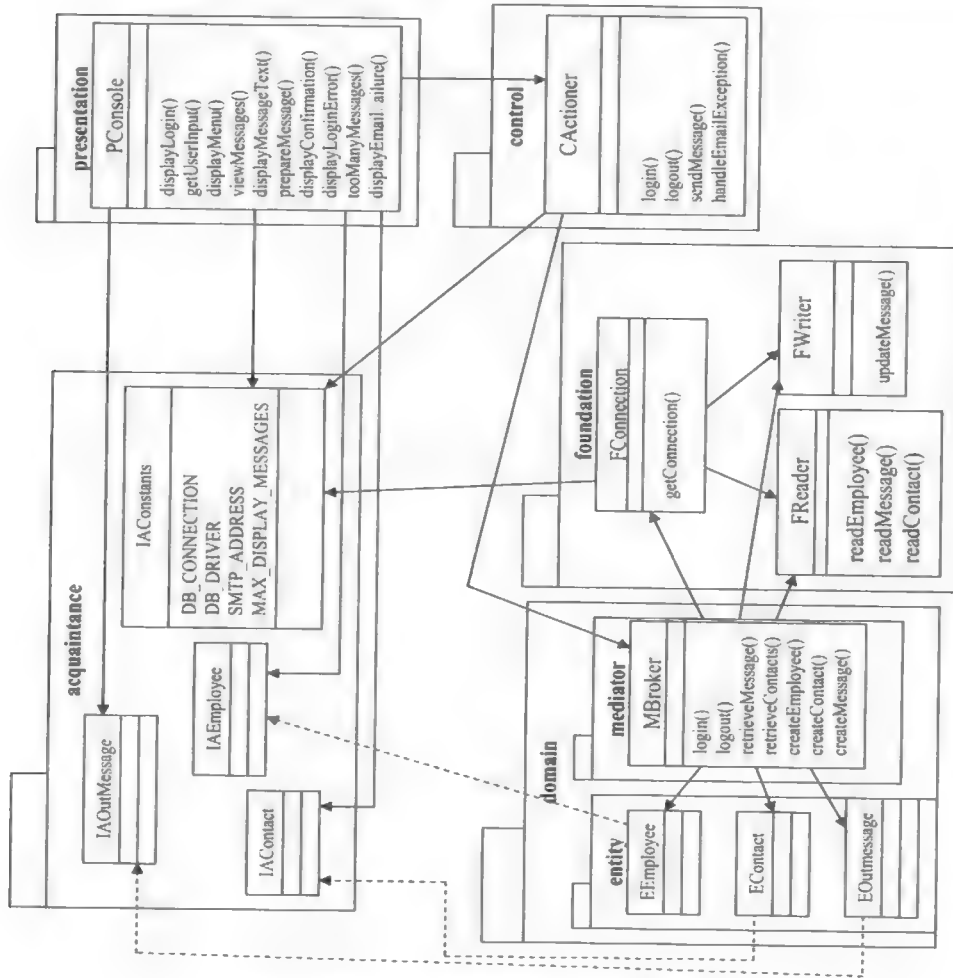
شكل (٨-٢) النموذج الأولي للأصناف للتطبيق EM

رقم المتطلب	الصنف/العملية	الحزمة/الصنف المتعاون	الإطار المعماري	الصنف/ العملية الجديد أو المعدل	الحزمة/الصنف المتعاون الجديد أو المعدل
R1	PConsole displayLogin PConsole getUserInput CActioner login	Presentation PConsole Control CActioner Mediator MBroker	DDP Chain of .Resp	MBroker login	Foundation FConnection
R2	FConnection connect FReader readEmployee	Acquaintance IAconstants Foundation FReader		MBroker createEmployee	entity EEmployee
R3	PConsole displayMenu				
R4	PConsole getUserInput CActioner exit	Control CActioner Mediator MBroker	DDP Chain of Resp	MBroker logout	Foundation FConnection
R5	PConsole getUserInput PConsole viewMessages MBroker retrieveMessages FReader readMessages FReader readContact	Presentation PConsole Acquaintance IAconstants	DDP Chain of Resp	CActioner retrieveMessages PConsole displayMessages MBroker retrieveContacts MBroker createContacts MBroker createMessages	Mediator MBroker Acquaintance IAEmployee IAOutMessage IAContact EEmployee EOutMessage EContact Foundation FReader entity EContact entity EOutMessage

Mediator MBroker Foundation FReader Acquaintance IAEmployee IAOutMessage IAContact EEmployee EOutMessage EContact	CActioner retrieveMessages MBroker isInCache PConsole displayMessageText	DDP Chain of Resp	Presentation PConsole	PConsole getUserInput PConsole displayMessageText	R6
Foundation FWriter	MBroker updateMessage MBroker flagCache	DDP Chain of Resp Mediator	Presentation PConsole Control CActioner Acquaintance IAconstants Foundation FWriter	PConsole getUserInput PConsole prepareMessage CActioner sendMessage PConsole displayConfirmation FWriter updateMessage	R7
Foundation FConnection	MBroker logout	DDP Chain of Resp	Presentation PConsole Ref. R1 Control CActioner	Ref. R1 PConsole displayLoginError CActioner exit	R8
Foundation FConnection	MBroker logout	DDP Chain of Resp	Presentation PConsole	PConsole getUserInput PConsole displayMenu CActioner logout ref. R1	R9
				Ref. R5 PConsole tooManyMessages	R10
			Presentation PConsole	Ref. R7 CActioner handleEmailException PConsole displayEmailFailure PConsole displayMenu	R11

شكل (٨-٣) توسعة جدول إيجاد الأصناف

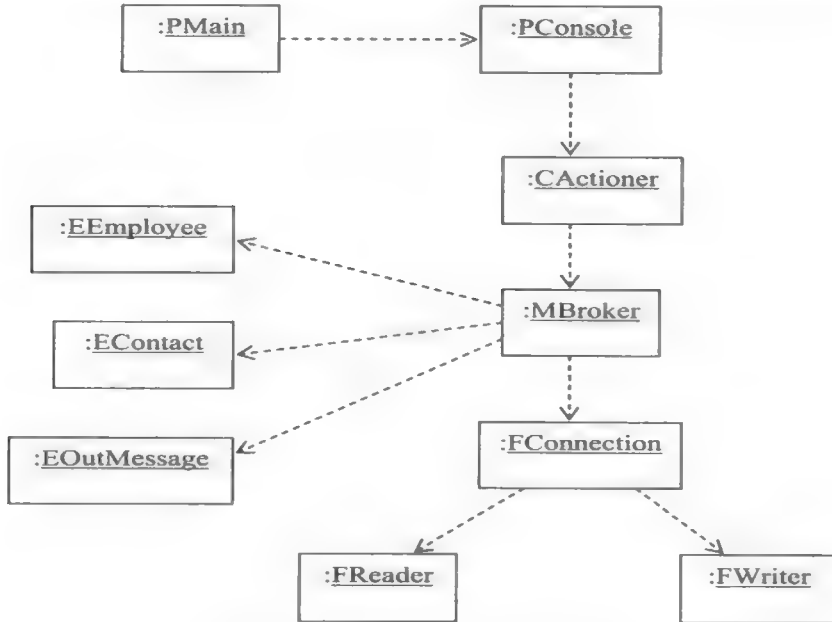
نلاحظ من الشكل أن الجدول يحتوى على ٦ أعمدة، الثلاثة الأولى منها مأخوذة من جدول إيجاد الأصناف (شكل (٨-١)) والثلاثة الأخيرة هي نتاج عملية التوسعة، حيث تم إضافة أصناف الكائنات entity classes وتوسعة الحزمة Acquaintance وإضافة عمليات للأصناف الموجودة في باقى الحزم. يوضح شكل (٨-٤) النسخة المطورة لنموذج الأصناف الخاص بالتطبيق EM بعد تطبيق الإضافات والتحسينات المذكورة في الجدول السابق (شكل (٨-٣)). نلاحظ من النموذج المطور إضافة حزمة entity إلى حزمة mediator لتكوين حزمة أشمل domain وهى تمثل إحدى طبقات الإطار المعماري PCMEF. ونلاحظ أيضا إضافة ثلاثة أصناف واجهة إلى حزمة Acquaintance يتم تطبيقها من خلال أصناف حزمة entity.



شكل (٨-٤) مخطط الأصناف للتطبيق EM

٨-٣ مخطط الكائنات Object Diagram:

يمثل الكائن إحدى حالات الصنف في وقت التنفيذ. يتم إنشاء الكائن الأول في التطبيق من خلال الطريقة الأساسية للبرنامج main الموجودة غالباً في حزمة الواجهة. يقوم هذا الكائن بإنشاء حالات أخرى للأصناف المرتبطة به طبقاً للتصميم المعماري لنموذج الأصناف وذلك من خلال تبادل الرسائل بين الكائنات. يوضح شكل (٨-٥) مخطط الكائنات للتطبيق EM والمقابل لمخطط الأصناف في الشكل السابق. نلاحظ من الشكل أن الرسائل بين الكائنات تتبع علاقات التبعية بين الأصناف وتعبّر الطبقات المعمارية من أعلى إلى أسفل حسب مواصفات التصميم المعماري PCMEF.



شكل (٨-٥) مخطط الكائنات للتطبيق EM

٨-٤ التفاعلات Interactions:

بعد الانتهاء من دراسة النظام في حالة السكون وتصميم نموذج الأصناف للنظام، ننقل الآن إلى دراسة النظام وهو في حالة الحركة أو ما يسمى بدراسة سلوك النظام System Behavior. تركز عملية السلوك على ملاحظة عمليات تبادل المعلومات بين

كائنات النظام فى أثناء تأدية وظيفة معينة وهو ما يطلق عليه اسم «التفاعلات بين الكائنات». يتم تحقيق هذه التفاعلات من خلال مجموعة من الرسائل Messages المتبادلة بين الكائنات. تعتبر الرسالة هى وسيلة اتصال بين كائن مرسل وآخر مستقبل للرسالة. تدعم لغة UML2 مجموعة من النماذج التى تستخدم لوصف أساليب التفاعلات بين كائنات النظام وهى مخطط التتابع Sequence Diagram ومخطط الاتصال Communication Diagram ومخطط التفاعل Interaction Diagram.

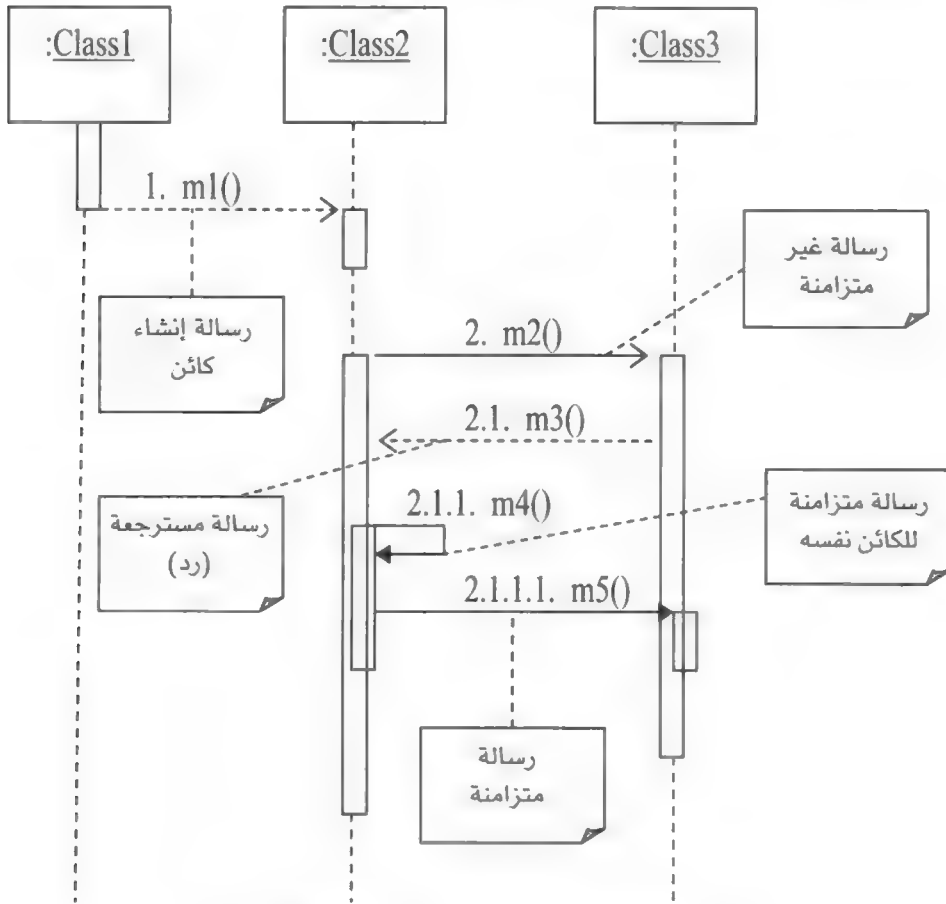
٨-٤-١ مخطط التتابع Sequence Diagram؛

يمثل مخطط التتابع مجموعة التفاعلات بين الكائنات داخل حالة استخدام معينة من خلال مجموعة الرسائل الممررة بين الكائنات. يتم عرض الكائنات فى أعلى المخطط ويتم رسم خط رأسى لكل كائن يمثل خط الحياة له Life Line. يتم تمثيل الرسالة المرسله من كائن إلى آخر بخط له سهم يتجه نحو الكائن المستقبل للرسالة. يتم رسم مستطيل ضيق رأسى حول خط حياة الكائن يمثل فترة تنشيط الكائن فى أثناء عملية تنفيذ العملية المطلوبة منه. يوضح هذا المخطط تتابع الرسائل بين الكائنات حيث إنه يهتم بوقت التنفيذ الموضح فى الاتجاه الرأسى. هناك نوعان من الرسائل: يمثل النوع الأول الرسائل المتزامنة Synchronous التى ينتظر فيها الكائن المرسل للرسالة حتى ينتهى الكائن المستقبل لها من تنفيذ الإجراءات المطلوبة. ويمثل النوع الثانى الرسائل غير المتزامنة Asynchronous وفيها يرسل الكائن الرسالة ولا ينتظر حتى ينتهى الكائن المستقبل لها من تنفيذها. يوضح شكل (٨-٦) أساليب تبادل الرسائل فى مخطط التتابع. يتم تمثيل الوقت فى مخطط التتابع فى الاتجاه الرأسى ويبدأ من أعلى متجهاً إلى أسفل. ويتم تمثيل الرسائل حسب الأسبقية، فتبدأ الرسائل الموجودة أعلى المخطط قبل الرسائل التى تليها لأسفل. وعلى هذا فإن مخطط التتابع يعتبر ثنائى الأبعاد، تمثل الكائنات الاتجاه الأفقى وتتجه من اليسار إلى اليمين، ويمثل الوقت الاتجاه الرأسى ويتجه من أعلى إلى أسفل. يلاحظ من الشكل أن أرقام الرسائل تشير إلى ترتيب تنفيذها وأنها ليست ضرورية، حيث إن خط الزمن يوضح عملية التتابع.

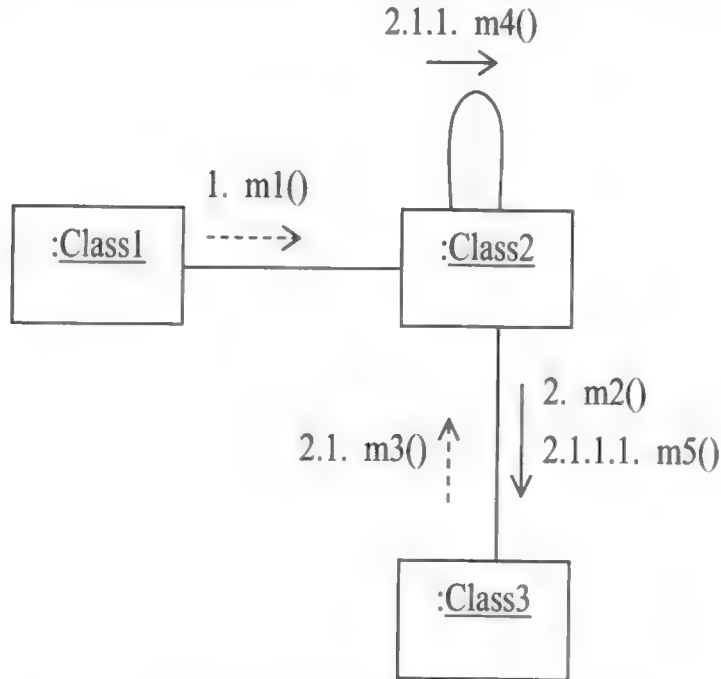
٨-٤-٢ مخطط الاتصال Communication Diagram؛

يعتبر مخطط الاتصال هو أحد أشكال مخططات التفاعل Interaction التى تقوم بتمثيل التفاعل بين الأشياء أو الكائنات. يعتبر مخطط الاتصال شبيهاً بمخطط

التتابع من ناحية تمثيل التفاعلات من خلال مجموعة الرسائل المتبادلة بين الكائنات. وجدير بالذكر أن معظم أدوات هندسة البرمجيات Case Tools تقوم بتحويل أحد المخططات إلى الآخر تلقائياً. ويتضح الفارق بينهما من ناحية تركيز مخطط التتابع على التسلسل الزمني للرسائل بين الكائنات، في حين يركز مخطط الاتصال على تمثيل العلاقات بين الكائنات بشكل واضح ومفهوم ولذلك فهو يستخدم في أثناء عملية تحليل الرسائل المرسله أو المستقبله لكائن ما، وتبادل وجهات النظر مع المستخدمين وباقي فريق العمل نظراً لسهولة بناء المخطط والتعديل فيه. يوضح شكل (٨-٧) مخطط الاتصال المقابل لمخطط التتابع الموضح بشكل (٨-٦).



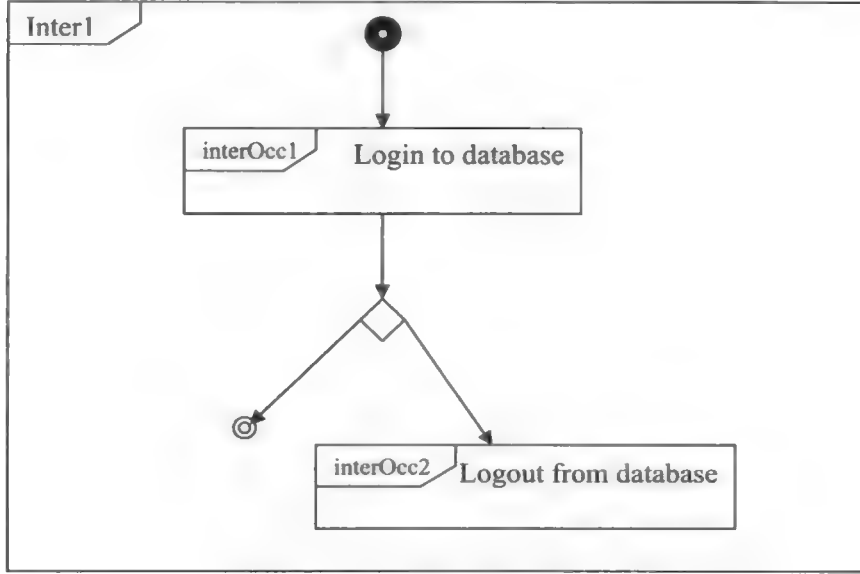
شكل (٨-٦) أنواع الرسائل في مخطط التتابع



شكل (٧-٨) تبادل الرسائل في مخطط الاتصال

٨-٤-٣ مخطط التفاعل Interaction Diagram

تعتبر مخططات التفاعل هي الأساس في نمذجة سلوك النظام باستخدام لغة UML. تتم عملية نمذجة السلوك في عدة مستويات من الإيجاز والتفصيل بدءاً من حالة استخدام أو مجموعة من حالات الاستخدام، إذ تتضمن بعض هذه التفاعلات بداخلها مجموعة من التفاعلات الداخلية. يؤدي ذلك إلى إمكانية الاستعانة بمخطط عام للتفاعلات يطلق عليه اسم «المنظور العام لمخطط التفاعل Interaction Overview Diagram» وهو يشبه إلى حد ما مخطط النشاط Activity diagram. يوضح شكل (٨-٨) مثالاً على المنظور العام لمخطط التفاعل الذي يمثل الانتقال بين التفاعلات الداخلية.



شكل (٨-٨) المنظور العام لمخطط التفاعل

٨-٥ تطبيق على التفاعلات Interactions Application

سنحاول في هذا القسم تصميم التفاعلات الخاصة بتطبيق إدارة البريد الإلكتروني EM. يحتوى هذا التطبيق على مجموعة محدودة من الأفعال Actions تتضمنها قائمة تشغيل التطبيق، بالإضافة إلى مجموعة الاستثناءات Exceptions أو رسائل الخطأ الموضحة في مواصفات الاستخدام. وفيما يلي وصف لمجموعة التفاعلات التي سيتم عرضها:

١- تفاعلات أساسية:

- الدخول Login.
- الخروج Exit.
- عرض الرسائل غير المرسل View Unsent Messages.
- عرض نص الرسالة Display Message Text.
- إرسال الرسالة Email Message.

٢- تفاعلات استثنائية؛

- كلمة مرور غير صحيحة Incorrect Username or Password.

- اختيار غير صحيح Incorrect Option.

- عدد كبير جداً من الرسائل Too Many Messages.

- عدم إمكانية إرسال الرسالة Email could not be sent.

جدير بالذكر أن هذا التطبيق لا يحتاج إلى المنظور العام لمخطط التفاعل ولكن سيتم استخدام مخططات التابع لوصف التفاعلات الأساسية ومخططات الاتصال لوصف التفاعلات الاستثنائية Exceptions.

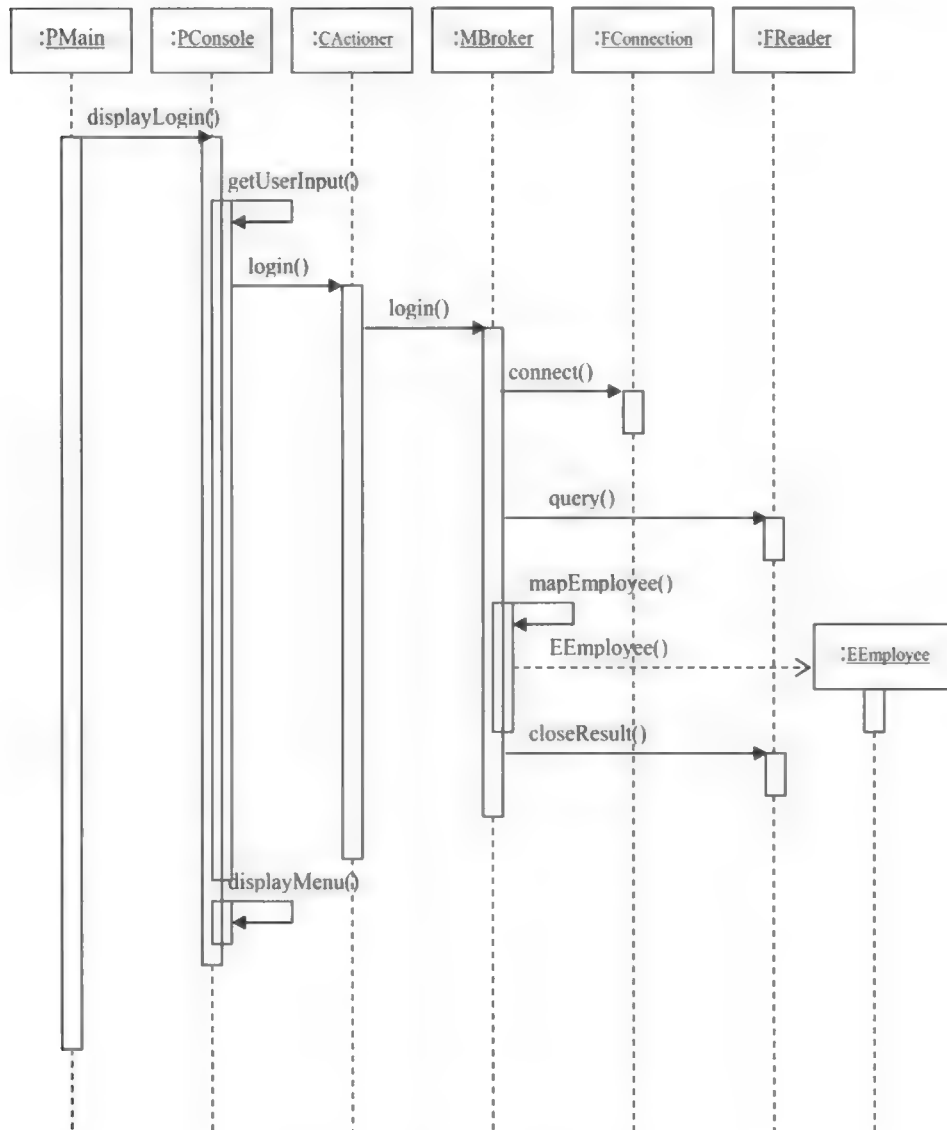
٨-٥-١ التفاعل «Login»:

تبدأ تفاعلات عملية الدخول للتطبيق بالرسالة () displayLogin من الكائن PMain إلى الكائن PConsole كما هو موضح في شكل (٨-٩). يقوم الكائن PConsole بعرض الرسالة () getUserInput للمستخدم للحصول على اسم المستخدم وكلمة المرور، ثم يقوم بتمريرها من خلال الرسالة () login إلى الكائن CActioner الذي يمررها بدوره إلى الكائن MBroker الذي يستخدمها في الاتصال بقواعد البيانات ومن ثم تكوين وتمرير الاستعلام () query إلى الكائن FReader الذي يقوم بتنفيذ الاستعلام والبحث عن بيانات الموظف في قواعد البيانات ويعود بالنتائج إلى الكائن MBroker الذي يقوم بتعبئة بيانات الكائن Employee، ثم يقوم بإغلاق عملية تنفيذ الاستعلام. وفي النهاية يقوم الكائن PConsole بعرض قائمة الاختيارات للمستخدم من خلال تنفيذ الطريقة () displayMenu.

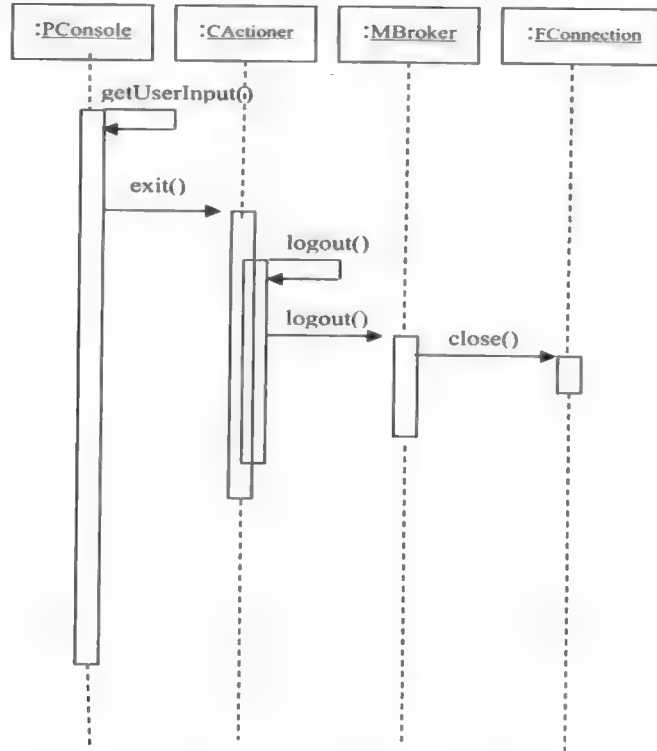
٨-٥-٢ التفاعل «Exit»:

يوضح الشكل (٨-١٠) مجموعة الرسائل التي تمثل التفاعل «Exit» في التطبيق EM وهي عبارة عن مجموعة من الرسائل الممررة بين الكائنات PConsole، FConnection لغلق قواعد البيانات، وكذلك رسالة إلى الكائن CActioner للخروج من النظام. تبدأ التفاعلات بتنفيذ الطريقة () getUserInput للكائن PConsole الذي يقوم بتمرير القيمة «خروج» إلى الكائن CActioner من خلال الرسالة () exit. يقوم الكائن CActioner بتنفيذ الطريقة () logout التي من خلالها يقوم بتمرير الرسالة logout

للكائن FConnection لغلق الاتصال بقواعد البيانات، وهو بدوره يمرر الرسالة Close () للکائن FReader لعملية القراءة من قواعد البيانات.



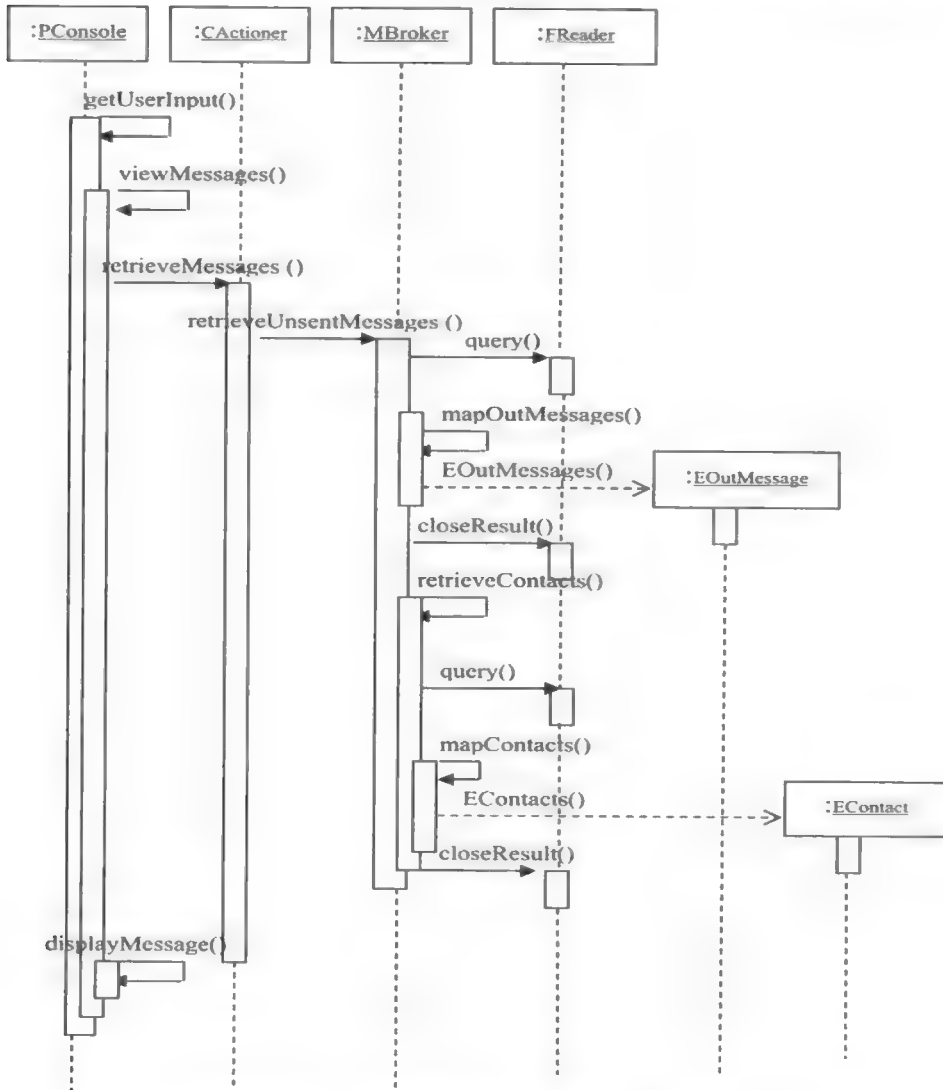
شكل (٩-٨) مخطط التفاعل ، Login



شكل (٨-١٠) مخطط التفاعل «Exit».

٨-٥-٣ التفاعل «View Unsent Messages».

يبدأ التفاعل بطلب المستخدم عرض الرسائل غير المرسلة من خلال الرسالة viewMessages للكائن PConsole الذي يطلب الخدمة retrieveMessages من الكائن CActioner الذي يمرره بدوره إلى الكائن MBroker مصحوباً بمعلومات الموظف المطلوب عرض رسائله. يقوم الكائن MBroker بطلب تنفيذ الاستعلام من الكائن FReader الذي يقوم بإنشاء كائن جديد EoutMessage ليتضمن مجموعة الرسائل الناتجة من الاستعلام. يتم استرجاع النتائج للكائن MBroker الذي يحتوي على بيانات جهة الاتصال لكل رسالة ويقوم بإنشاء كائنات لكل جهة اتصال EContact. وأخيراً يقوم الكائن PConsole بعرض الرسائل وجهات الاتصال باستخدام الخدمة displayMessage كما هو موضح بالشكل (٨-١١).



شكل (٨-١١) مخطط التفاعل «View Unsent Messages»

٨-٥-٤ التفاعل «Display Message Text»

من المفترض أن يبدأ هذا التفاعل بعد إتمام التفاعل السابق وفيه يقوم الكائن MBroker بالتحقق من وجود مجموعة الرسائل في الذاكرة من خلال الكائن

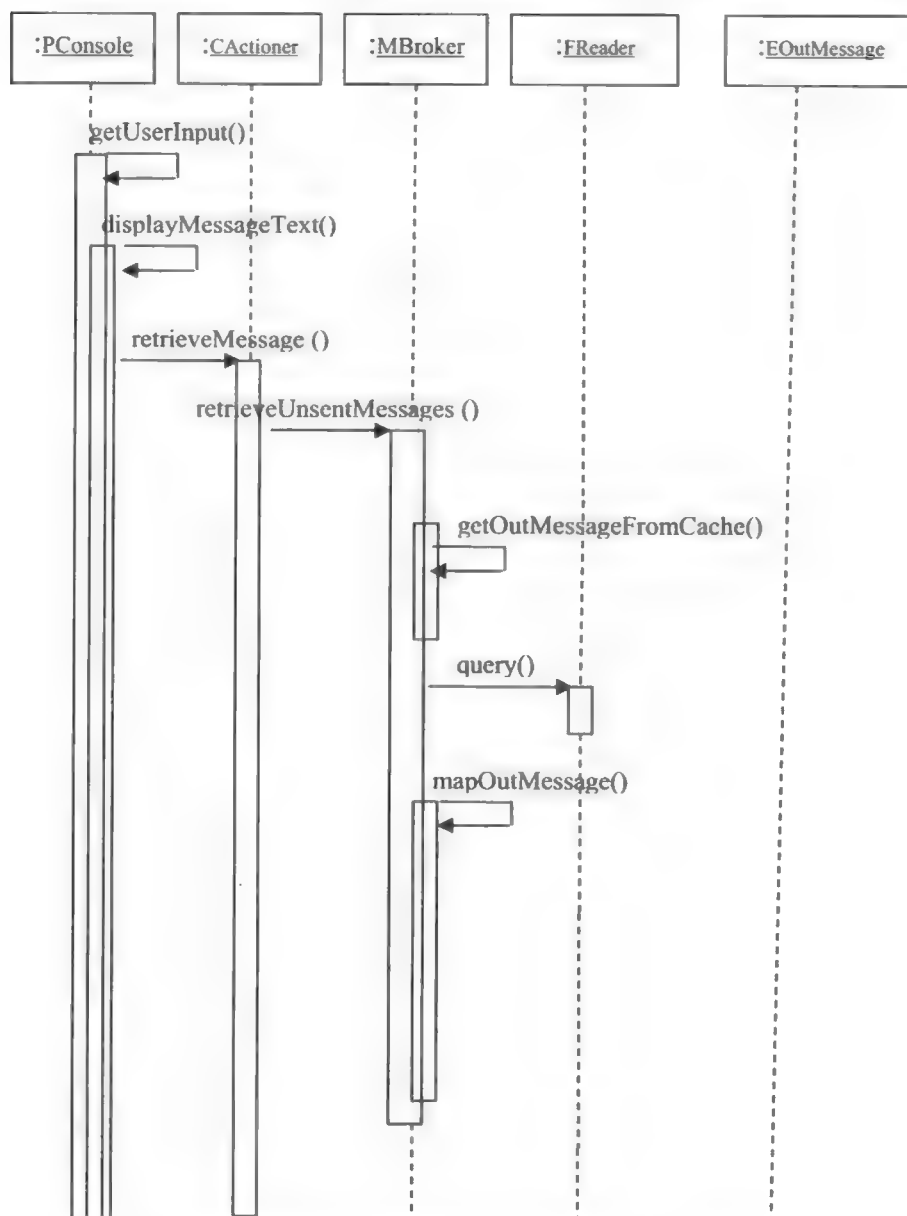
EoutMessage. وإذا لم يكن موجوداً يقوم الكائن بتكوين الاستعلام (query) وإرساله إلى الكائن FReader كما سبق شرحه. وبعد ذلك يقوم الكائن MBroker باسترجاع الرسائل إلى الكائن PConsole الذي يقوم بعرض نص الرسالة كما هو موضح بالشكل (٨-١٢).

٨-٥-٥ التفاعل «Email Message»

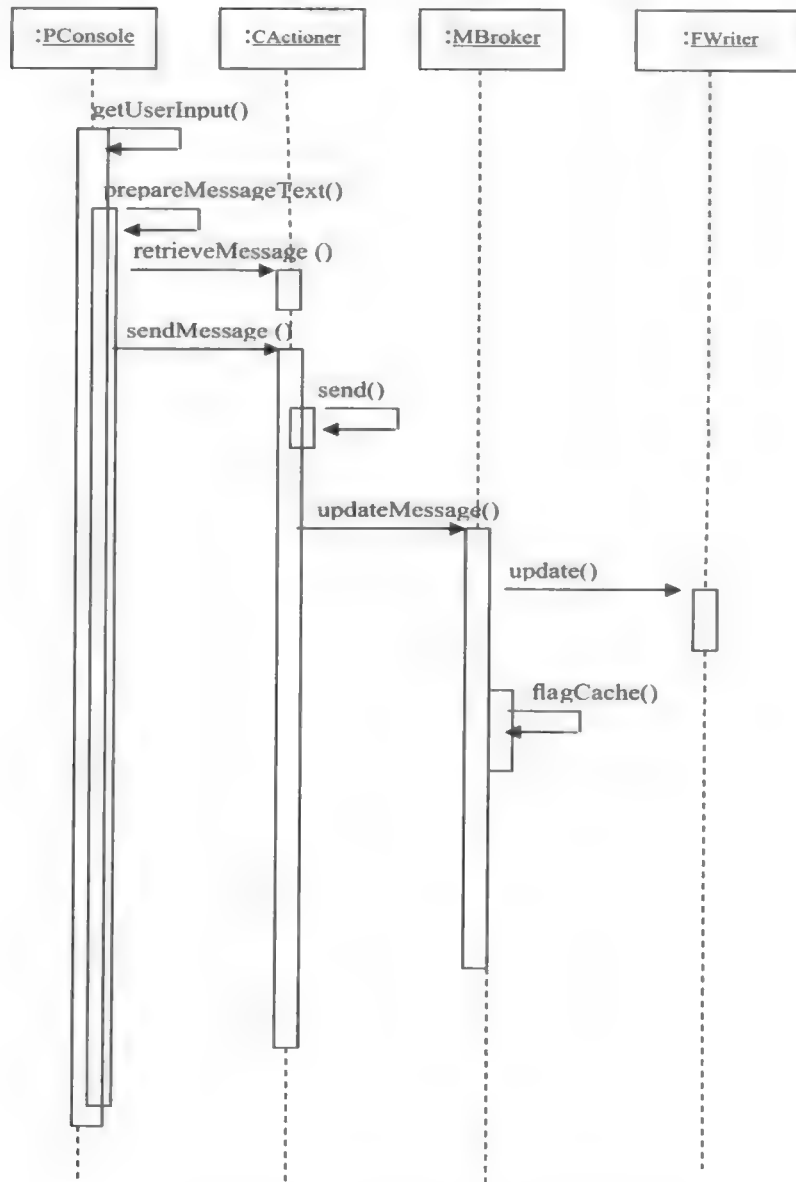
يبدأ التفاعل عندما يختار المستخدم أن يرسل رسالة تحتوي على نص موجود في قاعدة البيانات. يقوم الكائن PConsole بتجهيز الرسالة باستخدام الطريقة (prepareMessage) التي تقوم بإحضار نص الرسالة مستخدماً الخدمة (retrieveMessage) كما سبق عرضه في التفاعل السابق. وبعد ذلك يقوم الكائن PConsole بطلب إرسال الرسالة (sendMessage) من الكائن CActioner الذي يقوم باستخدام الطريقة (send). وبعد نجاح عملية الإرسال تقوم الطريقة (sendMessage) بتنشيط الطريقة (updateMessage) عند الكائن MBroker الذي يمرر جملة SQL إلى الكائن FWriter لتحديث قاعدة البيانات كما هو موضح بالشكل (٨-١٣).

٨-٥-٦ التفاعل «Incorrect User Name or Password»

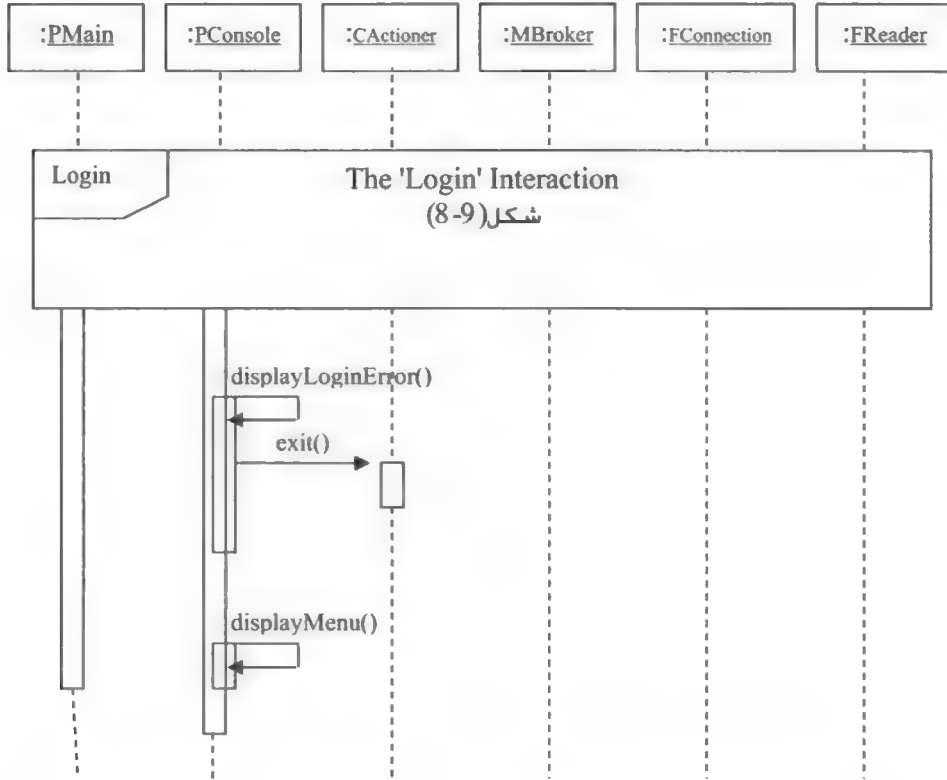
يبدأ هذا التفاعل في أثناء تنفيذ التفاعل «Login»، فعندما يسترجع الكائن PConsole القيمة «Unsuccessful Login» من تنفيذ الخدمة (getUserInput)، عندئذٍ يستخدم الكائن الطريقة (displayLoginError) لإظهار رسالة الخطأ للمستخدم، ثم يستخدم الطريقة (displayMenu) لعرض قائمة الاختيارات للمستخدم. إذا كانت هذه هي المرة الثالثة للخطأ فيقوم الكائن PConsole بإرسال الرسالة (exit) إلى الكائن CActioner للخروج من النظام، ثم يعرض القائمة الرئيسية displayMenu كما هو موضح بالشكل (٨-١٤).



شكل (٨-١٢) مخطط التفاعل «Display Message Text»



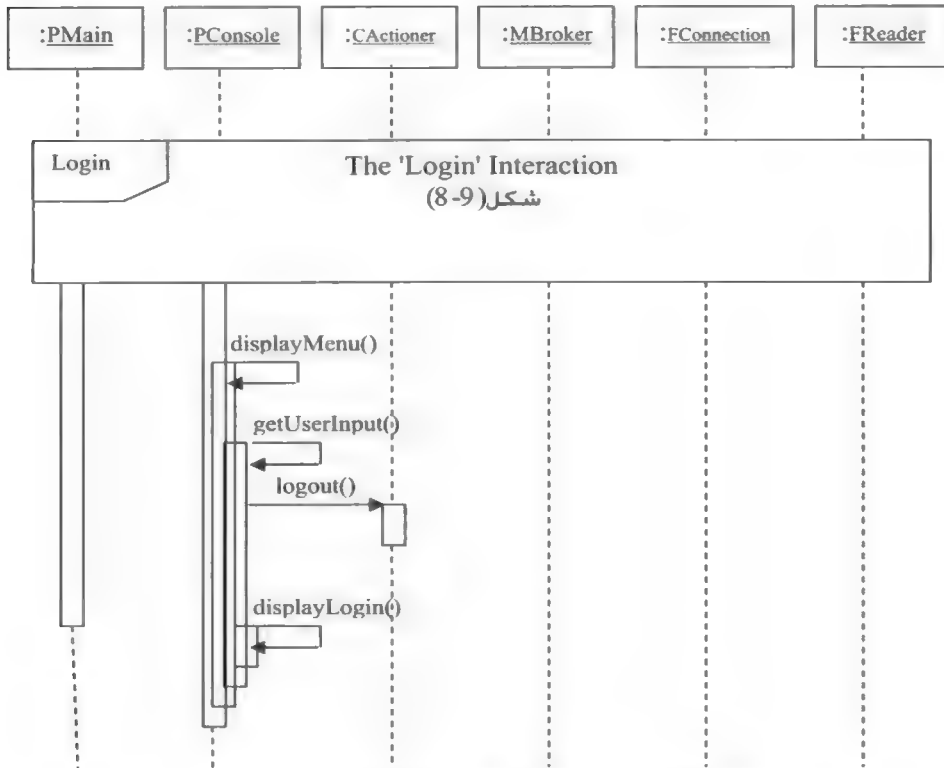
شكل (٨-١٣) مخطط التفاعل «Email Message»



شكل (٨-١٤) مخطط التفاعل «Incorrect User Name or Password»

٧-٥-٨ التفاعل «Incorrect Option»

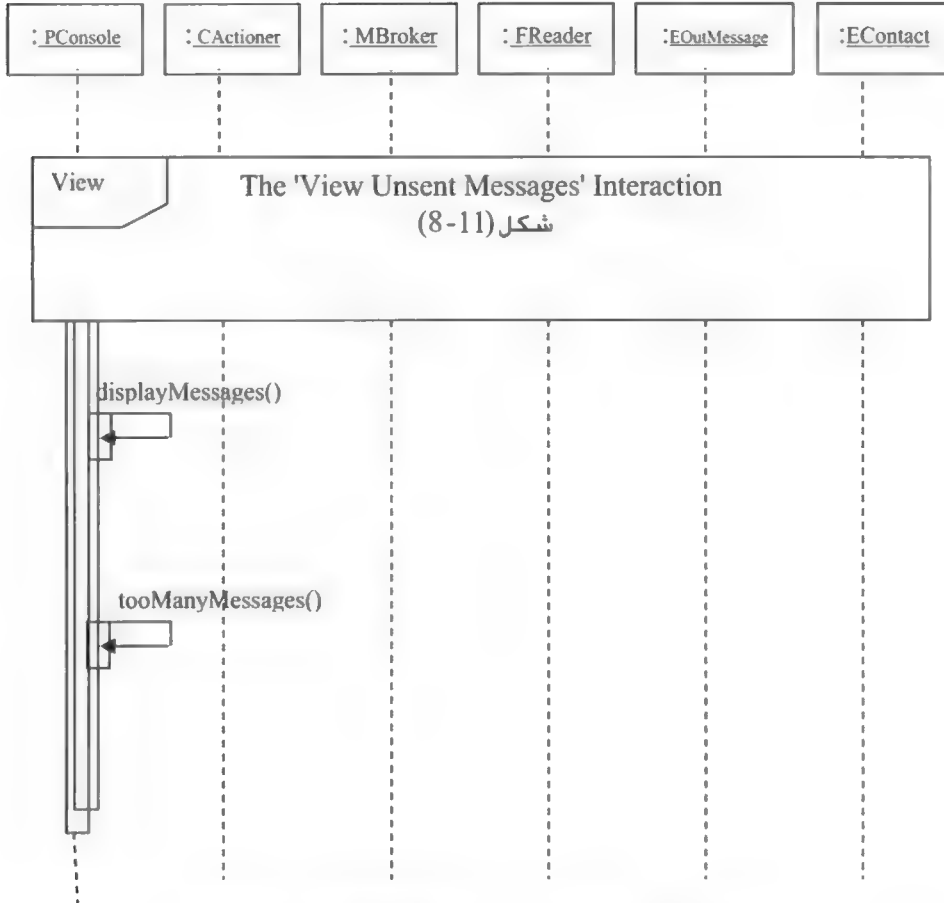
في حالة نجاح عملية الدخول «Login»، يقوم الكائن PConsole باستخدام الطريقة displayMenu () لعرض اختيارات القائمة للمستخدم. وبعد ذلك يستخدم الطريقة getUserInput () لقراءة رقم الاختيار المدخل من المستخدم. إذا كان الاختيار المدخل غير صحيح يتم إعادة عرض قائمة الاختيارات للمستخدم مرة أخرى. وفي حالة تكرار الخطأ ثلاث مرات متتالية يقوم الكائن PConsole باستدعاء الطريقة logout ()، ثم تعرض شاشة «Login» مرة أخرى، كما هو موضح بالشكل (٨-١٥).



شكل (٨-١٥) مخطط التفاعل «Incorrect Option»

٨-٥-٨ التفاعل «Too Many Messages»:

يستخدم الثابت Max - Display - Messages الموجود في الصنف IAContacts (شكل ٨-٤)) للتحكم في أقصى عدد للرسائل التي يمكن أن يعرضها البرنامج على شاشة المستخدم. يعتبر هذا التفاعل امتداداً للتفاعل «View Unsent Messages»، إذ يتم استدعاء الطريقة () retrieveMessages مصحوبة بمعامل يمثل عدد الرسائل numMsgs ويتم مقارنته بالثابت Max - Display - Messages. إذا كان العدد numMsgs أكبر من المسموح به فيتم عرض عدد الرسائل المسموح بها فقط باستخدام الطريقة () displayMessages ويتبعها تنفيذ الطريقة tooManyMessages لإعلام المستخدم بوجود عدد آخر من الرسائل المتبقية في قاعدة البيانات ولم يتم عرضها كما هو موضح بالشكل (٨-١٦).



شكل (٨-١٦) مخطط التفاعل «Too Many Messages»

٨-٥-٩ التفاعل «Email Could Not be Sent»

يبدأ هذا التفاعل في أثناء تنفيذ التفاعل «Email Message» عند حدوث استثناء في أثناء عملية الإرسال نتيجة عدم إتمامها بنجاح، ويتم استدعاء الطريقة `handleEmailException`، ويقوم الكائن `PConsole` باستدعاء الطريقة `displayEmailFailure` لإعلام المستخدم بفشل عملية إرسال الرسالة. وبعد ذلك يقوم بإعادة عرض قائمة الاختيارات كما هو موضح بالشكل (٨-١٧)، مع عدم تنفيذ التفاعل الخاص بتحديث قاعدة البيانات.

الفصل التاسع

البرمجة والاختبار

Programming and Testing

من المعروف أن الغرض الأساسي من عملية تطوير النظم هو إنتاج برمجيات قادرة على تحقيق متطلبات المستخدم. تعتبر عمليات البرمجة والاختبار هي المسؤولة عن إنتاج تلك البرمجيات حسب المواصفات التي تم تحديدها في أثناء تحليل المتطلبات وتصميم الحلول المناسبة لها. يتم اختبار البرمجيات المنتجة للتأكد من مطابقتها لمواصفات المتطلبات وبالجودة المطلوبة. ومن ثم فإن عمليات الاختبار تلازم عمليات البرمجة وأحياناً تكون عمليات البرمجة تابعة لعمليات اختبار محددة Test-Driven Programming. يقوم فريق المبرمجين بتطوير برامج الاختبار لكل وحدة برمجية منتجة، وهي عملية تتطلب وقتاً وجهداً كبيرين وتحتاج إلى تكلفة كبيرة. وفي النهاية يتم إجراء اختبار القبول Acceptance Test للنظام للتأكد من تحقيقه لجميع متطلبات المستخدم. وجدير بالذكر أنه يتوافر حالياً مجموعة كبيرة من لغات البرمجة وأدوات هندسة البرمجيات CASE Tools التي تستخدم في برمجة واختبار النظم البرمجية. يعود الاختلاف بين هذه اللغات والأدوات إلى الأساليب والمنهجيات التي تدعمها في أثناء عمليات التطوير. وسوف نقدم الآن فكرة عامة عن لغة جافا JAVA التي تعتبر من أكثر اللغات استخداماً في تطوير التطبيقات الحديثة نظراً لدعمها لأسلوب البرمجة المبنية على الكائنات Object-Oriented Programming الذي يلائم أسلوب تحليل وتصميم النظم باستخدام لغة UML، بالإضافة إلى دعمها القوى لتقنية برمجة التطبيقات التي تعمل على الويب.

٩-١ فكرة عامة عن لغة جافا Overview of JAVA Language:

سنحاول في هذا الجزء تقديم عرض سريع للسمات الأساسية للغة جافا وذلك من وجهة نظر هندسة البرمجيات. يركز العرض على المفاهيم الأساسية للغة جافا مثل الأصناف والعلاقات بينها والاتصال بقواعد البيانات التي تستخدم في تطوير معظم التطبيقات. وسوف نقوم بالاستعانة بالتطبيق الخاص بالتعامل مع قاعدة بيانات الأفلام Movie Actors التي تم تصميمها في الفصل السابع من الكتاب وذلك لتوضيح المفاهيم الأساسية للغة جافا.

٩-١-١ الأصناف Classes:

يمثل الصنف فى لغة جافا قالب الذى يتم إنشاء كائنات جديدة منه. وهو يحتوى على مجموعة من المحددات التى تصف المفاهيم التالية:

- اسم الصنف ومجال استخدامه Visibility.
- بيانات الكائن Data Members.
- العمليات التى يقوم بها الكائن Methods.
- الوحدات البنائية Constructors.
- مجال استخدام أعضاء الكائن (البيانات والعمليات).
- الصنف العلوى له Super class (إن وجد).
- واجهات الصنف Interfaces (إن وجدت).

تحدد خاصية Visibility مجال استخدام كل من كائنات وخصائص الصنف. غالباً ما يتم تعريف مجال الصنف على أنه عام Public لتتمكن باقى الكائنات من التعامل مع حالات الصنف Class Instances. ويتم تعريف معظم بيانات الصنف Class Data Members على أنها خاصة Private حتى يقتصر التعامل معها من خلال عمليات الصنف نفسه، فى حين يتم تعريف معظم عمليات الصنف على أنها عامة Public حتى يتم التعامل معها من خلال كائنات أصناف أخرى. يتم تعريف كل من متغيرات الصنف Class Variables وطرق الصنف Class Methods باستخدام الكلمة المحجوزة Static.

تدعم لغة جافا أسلوب التحميل الزائد للطرق Method Overloading، إذ تسمح لأكثر من طريقة داخل الصنف باستخدام الاسم نفسه ولكن مع اختلاف فى عدد المتغيرات الممررة Parameters للطريقة ونوعها. تدعم لغة جافا أيضاً أسلوب «نموذج الطريقة Method Prototype» الذى يتكون من اسم الطريقة والمعاملات الخاصة بها وأنواعها ونوع الناتج للطريقة ولكن بدون محتوى. يستخدم هذا النوع من الطرق لتعريف أصناف الواجهة Interface Classes.

يحتوى الصنف على نوع خاص من الطرق يسمى Constructor وهو يأخذ اسم الصنف نفسه ويستخدم فى إنشاء حالات الصنف (الكائنات الجديدة للصنف Objects). يمكن أن تأخذ هذه الطريقة عدة أشكال Overloading وذلك لإنشاء كائنات جديدة بطرق مختلفة. يوضح شكل (٩-١) تصميم الصنف Movie المقابل لجدول الأفلام. يحتوى الصنف على ثلاثة متغيرات Data Members، وطريقة إنشاء الكائنات Movie Constructor، ومجموعة من الطرق Methods التى تقوم بتخزين واسترجاع بيانات الكائن () Get و () Set.

Movie
movieTitle :String movieCode :double director :String
addListedAs(L: ListedAs): void removeListedAs(L: ListedAs): void getMovieTitle(): String getMovieCode(): double getDirector(): String setMovieTitle(): String setMovieCode(): double setDirector(): String equals(o:Object): boolean

شكل (٩-١) تصميم الصنف Movie

يتم استخدام هذا التصميم فى تطبيق الصنف باستخدام لغة جافا. يوضح شكل (٩-٢) محتوى البرنامج الذى يقوم بتطبيق الصنف Movie.


```

Movie.java

Public class Movie {
    Private String movieTitle;
    Private double movieCode;
    Private String director;
    Private Collection ListedAs;

    Public Movie(double movieCode, String title, String director) {
        This.movieCode = movieCode;
        This.director = director;
        This.movieTitle = title;
        ListedAs = new ArrayList();
    }

    Public void addListedAs(ListedAs l) {
        ListedAs.add(l);
    }

    Public void removeListedAs(ListedAs l) {
        ListedAs.remove(l);
    }

    Public String getMovieTitle() {
        Return movieTitle;
    }

    Public void setMovieTitle(String title) {
        This.movieTitle = title;
    }

    Public double getMovieCode() {
        Return movieCode;
    }

    Public void setMovieCode(double code) {
        This.movieCode = code;
    }

    Public String getDirector() {
        Return director;
    }

    Public void setDirector(String director) {
        This.director = director;
    }

    Public Boolean equals(Object o) {
        Try {
            Movie m = (Movie) o;
            If m.movieCode == movieCode ) return true;
        } catch(Exception exc) {}
        Return false;
    }
}

```

شكل (٢-٩) محتوى البرنامج Movie.java

٢-١-٩ ارتباط الأصناف Class Associations:

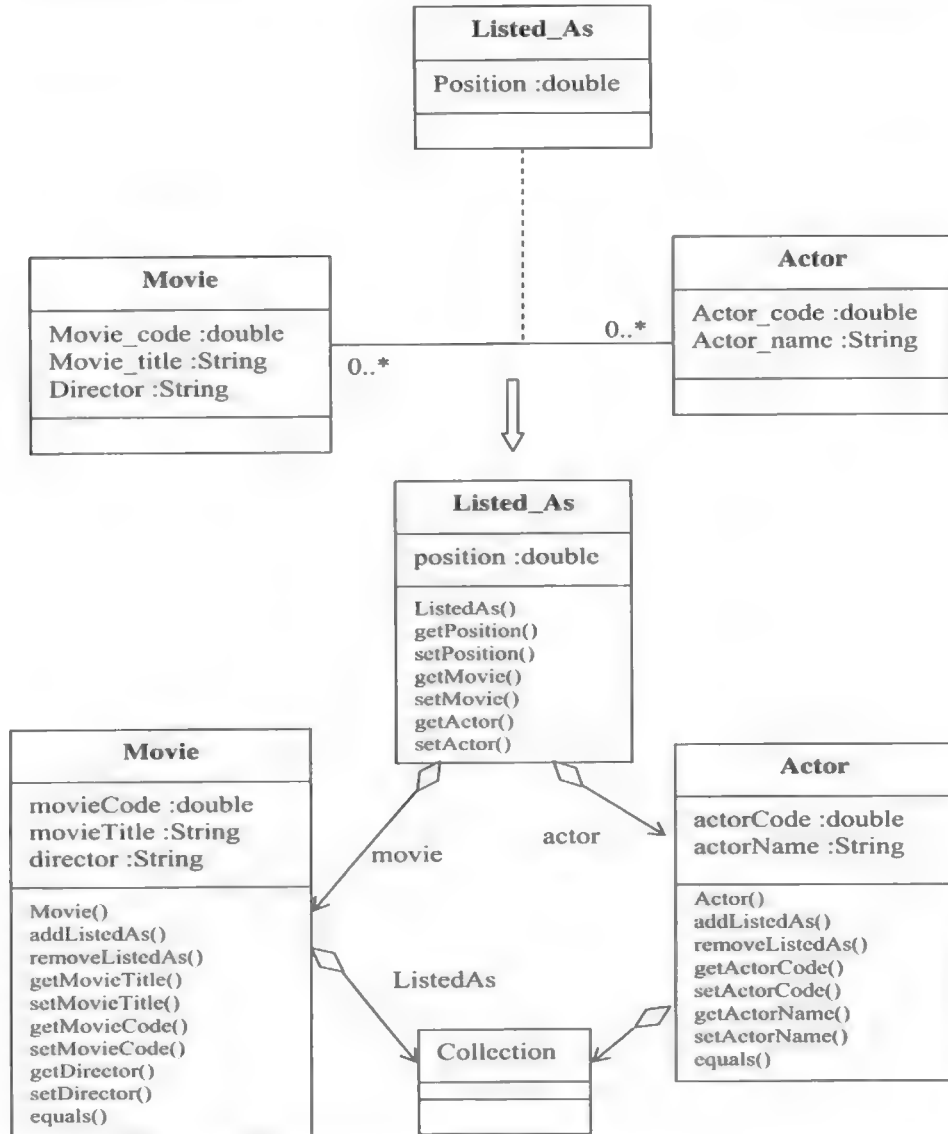
يجب الاهتمام بتعريف مسارات الاتصال بين الكائنات بوضوح تام في البرنامج التطبيقى مما يؤدى إلى فهم البرنامج وسهولة صيانه وتوسعته فيما بعد . تسهم

عملية الاتصال بين الأصناف من خلال الرسائل فى توضيح العلاقات بينها . يتم تمثيل الارتباط بين الأصناف من خلال متغيرات الحالات Instance Variables التى تخزن معلومات تشير إلى الكائنات المرتبطة . يتم استخدام متغير حالة واحد فى حالة الارتباط بكائن واحد فقط ، ويتم استخدام مجموعة متغيرات تسمى «Collection» فى حالة الارتباط بأكثر من كائن . ولتوضيح ذلك سوف نستعين بنموذج الأصناف فى تطبيق الأفلام يصف علاقة الارتباط من النوع متعدد-إلى-متعدد بين الصنف «Actor» والصنف «Movie» عن طريق صنف ثالث (صنف الارتباط Association Class) وهو صنف «Listed-As» كما هو موضح بالشكل (٩-٣).

ويوضح الشكل أيضاً كيفية تحويل هذا النموذج إلى النموذج الذى سيتم تطبيقه باستخدام لغة جافا . يحتوى النموذج التطبيقي على ثلاثة أصناف تمثل كائنات كل من «Listed-As» ، «Actor» ، «Movie» ، ويحتوى كل صنف على جميع الخصائص Attributes والطرق Methods الخاصة به . نلاحظ من الشكل أيضاً أنه لم يتم تمثيل العلاقة بين «Actor» و«Movie» من خلال علاقته ربط من النوع واحد - إلى - متعدد كما هو الحال فى مخطط الأصناف ، ولكن تم تمثيل العلاقة من خلال أربع علاقات ربط أحادية الاتجاه وأحادية التعددية ومتضمنة أسلوب المجموعات Collections الخاص بلغة جافا الذى سنوضحه الآن بشئ من التفصيل .

مجموعات جافا Java Collections :

على النقيض من قواعد البيانات المعتمدة على الكائنات ، فإن لغات البرمجة المعتمدة على الكائنات لا تدعم بشكل مباشر عملية تطبيق علاقات الربط Associations بين الكائنات . ولكنها تعود إلى مهارة المبرمج فى تمثيل العلاقات بين الكائنات من خلال مجموعة من الأصناف التى تتضمن معلومات عن علاقات الربط المطلوبة . وتزداد هذه العملية صعوبة فى حالة العلاقات ثنائية الاتجاه Bidirectional Associations . وفى حالة العلاقات أحادية الاتجاه التى يشارك فيها كائن واحد فقط يتم تمثيل العلاقة من خلال متغير حالة Instance Variable يأخذ نوع الصنف ، وتكون قيمة هذا المتغير فى شكل إشارة Reference إلى كائن من هذا الصنف .



شكل (٩-٣) تحويل نموذج الأصناف إلى النموذج التطبيقي

وفى حالة مشاركة أكثر من كائن واحد فى العلاقة يتم تمثيل العلاقة من خلال متغير حالة يأخذ نوع واجهة مجموعة Collection Interface مأخوذة من مكتبة جافا

java.util.collection. وتكون قيمة المتغير في شكل إشارة إلى بعض الكائنات التي تطبق هذه الواجهة. يوضح شكل (٩-٤) واجهة مجموعة Collection Interface.

Collection
<pre> add(obj: Object): Boolean addAll(collection: Collection): Boolean clear(): void contains(obj: Object): Boolean containsAll(collection: Collection): Boolean equals(obj: Object): Boolean hashCode(): int isEmpty(): Boolean iterator(): iterator remove(obj: Object): Boolean removeAll(collection: Collection): Boolean retainAll(collection: Collection): Boolean size(): int toArray(): Object[] toArray(aobj[]: Object): Object[] </pre>

شكل (٩-٤) واجهة مجموعة Collection Interface

تعتبر واجهة المجموعة بمنزلة مجمع للكائنات وهو يتضمن مجموعة من الطرق التي تسمح لها بإضافة كائنات جديدة أو حذف كائنات منها وكذلك البحث عن وجود كائن معين والتحقق من عدد الكائنات الموجودة بالمجمع. تحتوى مكتبة مجموعات جافا java.util.collection على ثلاث مجموعات هي List و Set و Map. يوضح شكل (٩-٥) برنامجاً بلغة جافا لتمثيل الصنف Actor الذي يتضمن تعريفاً للمتغير ListedAs من النوع مجموعة Collection الذي يشير إلى مجموعة كائنات ListedAs الموضحة في شكل (٩-٣). يتضمن البرنامج أيضاً الطريقة AddListedAs لإضافة كائنات جديدة للمجموعة وطريقة RemoveListedAs لحذف كائنات من المجموعة ListedAs. ويوضح شكل (٩-٦) برنامج جافا لتمثيل الصنف ListedAs الذي يبين كيفية قيام الكائن ListedAs بتأسيس العلاقة بين الكائن Actor والكائن Movie. يتم استخدام إشارة إلى الكائنين في شكل معاملات تمرر إلى الطريقة البنائية للكائن

ListedAs. وهنا يقوم الكائن ListedAs بإضافة نفسه كعامل يمرر إلى الطريقة AddListedAs () لكل من الكائن Movie والكائن Actor، مما يؤدي إلى ضمان السلامة المرجعية بين الكائن ListedAs والكائنين Actor و Actor.

Actor.java

```
Public class Actor {

    Private double actorCode;

    Private String actorName;

    Private Collection listedAs;

    Public Actor(double code, String name) {
        this.actorCode = code;
        this.actorName = name;
        listedAs = new ArrayList();
    }

    Public void addListedAs(ListedAs l) {
        listedAs.add(l);
    }

    Public void removeListedAs(ListedAs l) {
        listedAs.remove(l);
    }

    Public double getActorCode() {
        Return actorCode;
    }

    Public void setActorCode(double code) {
        This.actorCode = code;
    }

    Public String getActorName() {
        Return actorName;
    }

    Public void setActorName (String actor) {
        This.actorName = actor;
    }

    Public Boolean equals(Object o) {
        Try {
            Actor a = (Actor) o ;
            If(a.actorCode == actorCode ) return true;
        } catch(Exception exc) {}
        Return false;
    }

}
```

شكل (٩-٥) تمثيل الصنف Actor بلغة جافا

ListedAs.java

```
Public class ListedAs {

    Private double position;

    Private Movie movie;

    Private Actor actor;

    Public ListedAs(Movie m, Actor a, double position) {
        this.movie = m;
        this.actor = a;
        this.position = position;

        movie.addListedAs(this);
        actor.addListedAs(this);
    }

}
```

شكل (٩-٦) تمثيل الصنف ListedAs بلغة جافا

٩-١-٣ الاتصال بقواعد البيانات Database Connections

يتم التعامل مع قواعد البيانات المرتبطة Relational Database باستخدام تعليمات لغة الاستفسار البنائية SQL. تحتوى معظم لغات البرمجة على مكونات خاصة بالاتصال والتعامل مع قواعد البيانات. تتضمن هذه المكونات داخلها تعليمات لغة الاستفسار البنائية. فعلى سبيل المثال نجد أن بيئة التطوير الخاصة بلغة جافا تحتوى على المكون (Java Database Connectivity (JDBC) الذى يمكن لغة جافا من الاتصال بقواعد البيانات والتعامل معها من خلال تعليمات SQL. تحتوى لغة جافا على مكونات أخرى تستخدم للتعامل مع قواعد البيانات مثل:

- Enterprise Java Beans (EJB).
- Business Components for Java (BC4J).

تدعم معظم أنظمة إدارة قواعد البيانات DBMS مثل «أوراكل» إمكانية أن تقوم لغات البرمجة مثل «جافا» بتخزين بعض الإجراءات في قواعد البيانات للتعامل السريع معها وذلك من خلال مجموعة من الإجراءات المخزنة Stored Procedures. يمثل الشكل (٧-٩) جزءاً من برنامج جافا للتعامل مع قواعد البيانات باستخدام تقنية JDBC. تتم عملية الاتصال من خلال الصنف connection والصنف DriverManager من مكتبة Java.SQL. يتم إنشاء كائن الاتصال conn الذي يقوم باستدعاء الطريقة getConnection () مستخدماً ثلاث معاملات تمثل مسار قاعدة البيانات المطلوب الاتصال بها، واسم مستخدم قاعدة البيانات، وكلمة السر للاتصال بقاعدة البيانات.

Connection to the Database

```
Public class Connection {

    Private java.sql.connection conn;

    Public connection () throws Exception {

        Class.forName("oracle.jdbc.driver.oracleDriver");
        Conn = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:oracle8i", "psel", "psel");
    }

}
```

شكل (٧-٩) الاتصال بقواعد البيانات في لغة جافا

يحتوي صنف واجهة الاتصال Connection Interface على ثلاثة طرق تستخدم لتنفيذ تعليمات SQL هي: Statement, preparedStatement, callableStatement. تستخدم Statement لتنفيذ تعليمات SQL البسيطة التي لا تحتوي على معاملات. وتستخدم prepared Statement لتنفيذ تعليمات SQL المتضمنة شروطاً أو معاملات للبحث في قواعد البيانات. وتستخدم callableStatement لاستدعاء وتنفيذ الإجراءات المخزنة في قواعد البيانات Stored Procedures.

ولتطبيق ذلك على قاعدة بيانات الأفلام فإنه يتطلب الاتصال بقاعدة البيانات والبحث فيها عن بيانات الكائنات وتحميلها في ذاكرة التطبيق. يوضح شكل (٨-٩) كائن البحث MovieSearcher الذي يقوم بإنشاء كائن اتصال Conn وبعد ذلك يقوم باستدعاء الطريقة () readAll. ويوضح الشكل (٩-٩) تفاصيل الطريقة () readAll، حيث يتم تكوين تعليمة SQL وتمريرها إلى الطريقة () query. وبعد ذلك يتم إنشاء الكائن st الذي يقوم بتنفيذ تعليمة SQL على قاعدة البيانات ويتم تخزين نتائج البحث في الكائن rs.

MovieSearcher.java

```
Public class MovieSearcher {

    Private Connection conn;
    Private Collection ListedAs;

    Public MovieSearcher() throws Exception {
        Conn = new Connection();
        readAll();
    }

    Public static void main(String args[]) {
        MovieSearcher searcher = new MovieSearcher();
    }
}
```

شكل (٨-٩) برنامج البحث بلغة جاوا


```

Connection.java
Public class Connection {

    Private java.sql.connection conn;

    Public connection () throws Exception {

        Class.forName("oracle.jdbc.driver.oracleDriver");
        Conn = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:oracleai", "pse1", "pse1");
    }

    Public Collection readAll() throws Exception {
        Collection c = new ArrayList();
        ResultSet rs = query("Select * from movie m, actor a, listed_as l
            Where m.movie_code = l.movie_code
            And l.actor_code = a.actor_code");

        While (rs.next()) {
            Double movieCode = rs.getDouble("movie_code");
            String movieTitle = rs.getString("movie_title");
            String director = rs.getString("director");
            Double actorCode = rs.getDouble("actor_code");
            String actorName = rs.getString("actor_name");
            Double position = rs.getDouble("position");

            Movie m = new Movie(movieCode, movieTitle, director);
            Actor a = new Actor(actorCode, actorName);
            ListedAs la = new ListedAs(m, a, position);
            c.add(la);
        }

        Return c;
    }

    Private ResultSet query(String sql) throws Exception {
        Statement st = conn.createStatement();
        Return st.executeQuery(sql);
    }
}

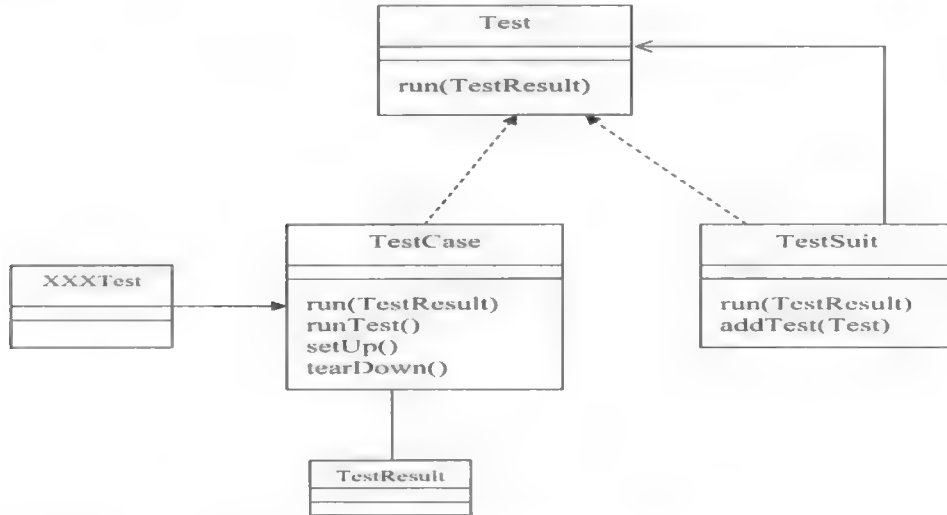
```

شكل (٩-٩) برنامج الاتصال وتنفيذ عملية البحث في قواعد البيانات

٩-٢ التطوير المبني على الاختبار Test-Driven Development:

تقوم بعض أساليب تطوير البرمجيات على فكرة الاختبار بشكل أساسى مثل الأسلوب المقدم من شركة Agile (Martin, 2003). يعتمد هذا الأسلوب على فكرة تجهيز برنامج الاختبار أولاً قبل البدء فى برمجة التطبيق. وعلى هذا فإنه يتم تطوير برامج التطبيق بفرض اجتياز الاختبارات المعدة مسبقاً مما يؤدى إلى ضمان كفاءة برامج التطبيق وتأديتها للوظائف المطلوبة بنجاح. تعتبر هذه العملية تكرارية، حيث يتم إعداد برنامج اختبار أولى لبعض الوظائف الأساسية بالنظام وتجهيز حالات الاختبار لها. وبعد ذلك يتم كتابة أكواد البرنامج المنفذ لتلك الوظائف وتنفيذه لاجتياز الاختبار المعد له مسبقاً. وعندما يجتاز البرنامج الاختبار يتم توسعة برنامج الاختبار الأولى بإضافة حالات اختبار أخرى لوظائف جديدة فى النظام، ويتم أيضاً توسعة البرنامج التطبيقي المقابل له، ويتم تنفيذه لاجتياز الاختبارات المعدة له وهكذا حتى

يتم الانتهاء من تطبيق جميع وظائف النظام. والسؤال الآن هو أين يتم وضع الكود الخاص بحالات الاختبار بالنسبة للكود الخاص بالتطبيق المطلوب اختباره؟ هناك مجموعة من الأساليب المتبعة لحل هذا السؤال. أحد هذه الأساليب هو وضع الكود الخاص بحالة الاختبار Test Case داخل الطريقة الرئيسية () main لكل صنف يتم اختباره. الأسلوب الثاني هو وضع حالة الاختبار في صنف داخلي inner Class داخل الصنف المطلوب اختباره. والسبب في ذلك هو أن الكائن الداخلي (كائن الاختبار) يمكنه أن يتعامل بحرية مع كل عناصر الكائن الخارجى له (الكائن تحت الاختبار). يوضح شكل (٩-١٠) إطاراً عاماً لنموذج الاختبار. يحتوى النموذج على صنفى اختبار أساسيين هما Test Case، TestSuit وكلاهما يكون امتداداً أو تطبيقاً لصنف واجهة الاختبار test الذى يصف الطريقة () run لتففيذ الاختبار. يعتبر الصنف TestSuit من الأصناف التى تحتوى على مكونات Composite Class وهو يمكن أن يتضمن أكثر من حالة اختبار، أى أن الكائن TestSuit يمكن أن يحتوى على عدة كائنات اختبار TestCase. أى أن هذا النموذج يسمح بحالات اختبار فردية من خلال الكائن TestCase ويسمح أيضاً بحالات اختبار مركبة من خلال كائنات TestSuit.



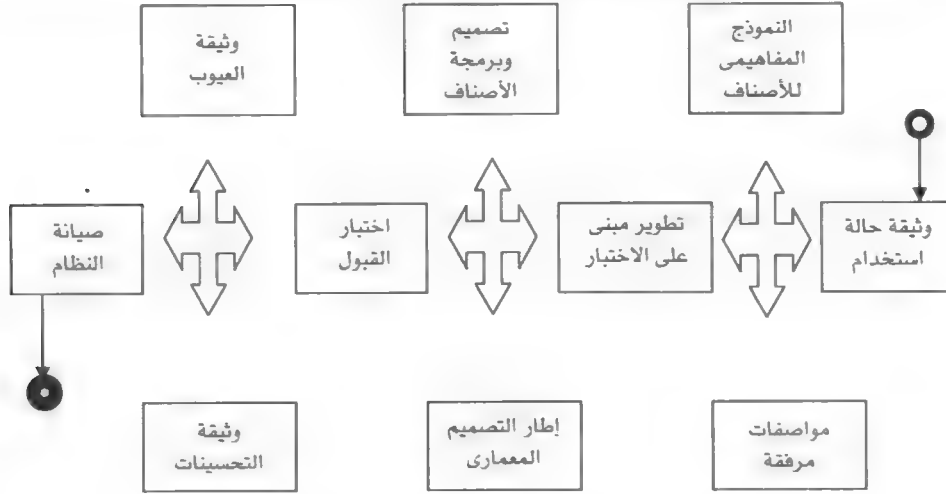
شكل (٩-١٠) إطار عام لنموذج الاختبار

يتم تطبيق وحدة الاختبار من خلال صنف فرعى هو امتداد للصنف TestCase. يأخذ هذا الصنف الفرعى الاسم xxxTest كما هو موضح بالشكل ويقوم بإضافة

طرق جديدة لاختبار الوحدة وتكون امتداداً للطرق الموجودة بالصنف TestCase. تقوم الطريقة () Setup للكائن TestCase بإنشاء كائنات وحدة الاختبار التي تمثل الهدف لكود الاختبار. وتقوم الطريقة () TearDown بإزالة كائنات وحدة الاختبار من الذاكرة بعد انتهاء كل حالة اختبار. وتقوم الطريقة () RunTest بتنفيذ وحدة الاختبار وذلك من خلال استدعاء حالة الاختبار المطلوبة داخل الكائن xxxTest.

على الرغم من أن عملية التطوير المبني على الاختبار تركز على اختبار وحدات صغيرة مثل الأصناف (كل صنف على حدة أو مجموعة بسيطة من الأصناف) فإنها لا تستطيع أن تكتشف الأخطاء في تطبيق المتطلبات الوظيفية للنظام العام أو تكتشف عدم إنجاز بعض المهام الوظيفية. وعلى هذا فإن اختبار الوحدات الصغيرة Units Test يسمى «اختبار الصندوق الأبيض White Box Test» لأنه يركز على كيفية تطبيق المهام أو الوظائف داخل كل صنف أو وحدة. أما النوع الآخر من الاختبار الذي يطلق عليه اسم «اختبار الصندوق الأسود Black Box Test» فهو يهتم بعملية التأكد من تحقيق المتطلبات الوظيفية للنظام بغض النظر عن كيفية تحقيقها ولكنه يركز على اكتمال المتطلبات ومطابقتها لمواصفات متطلبات النظام. وعلى هذا فإن هذا النوع من الاختبار يتم بواسطة العميل Customer والمستخدم User للنظام ولذلك فهو يسمى «اختبار القبول Acceptance Test» من ناحية المستفيد. يوضح شكل (٩-١١) دور عمليات الاختبار في دورة حياة تطوير التطبيقات. تبدأ عملية التطوير المبنية على الاختبار بمجموعة من وثائق حالات الاستخدام Use Case Documents والنموذج المفاهيمي للأصناف Conceptual Class Model وبعض المواصفات الإضافية. وبعد ذلك يتم تجهيز حالات الاختبار Test Cases التي سيتم استخدامها في بناء برامج وحدات الاختبار Test Units. وبعد انتهاء عمليات تطوير واختبار وحدات التطبيق يتم تنفيذ اختبار القبول على التطبيق النهائي.

ولأن اختبار القبول لا يهتم بكيفية التطبيق فإنه يتم تجهيزه بشكل منفصل عن عملية التطبيق وذلك من خلال مجموعة المستخدمين والمستفيدين من النظام. يتم تجهيز مواصفات الاختبار في شكل مجموعة من سيناريوهات الاختبار Test Scripts. يتم تسجيل مجموعات العيوب الناتجة عن اختبار القبول في شكل وثيقة العيوب Defects Document ليتم تصحيحها من قبل المطورين للتطبيق. ويتم تسجيل مجموعة الملاحظات حول التحسينات المطلوبة في شكل وثيقة التحسينات Enhancements Document.



شكل (٩-١١) دور الاختبار في تطوير التطبيقات

ويتم تطبيق تلك التحسينات وتصحيح العيوب من خلال عمليات الصيانة للتطبيق. يتضمن اختبار القبول مجموعة من نقاط التحقق Verification Points التي تستخدم في تحديد النتائج المتوقعة من ناحية المستفيد بالنسبة لسلوك النظام الوظيفي مقابل مجموعة المدخلات والشروط وقواعد العمل المتضمنة في الوظيفة تحت الاختبار. يتضمن اختبار القبول مواصفات البيانات التي سيتم استخدامها في عملية الاختبار والنتائج المتوقعة المقابلة لتلك البيانات. يوضح شكل (٩-١٢) محتوى وثيقة سيناريو اختبار القبول لتطبيق إدارة البريد الإلكتروني EM. تحتوى الوثيقة على ثلاثة أعمدة: الأول يصف رقم الخطوة (S) أو نقطة التحقق (V) Verification Point. والعمود الثانى يصف النتيجة المتوقعة من التحقق True, False, Unknown. من الممكن أن تكون نتيجة التحقق فى الحالات الأولية «غير معروفة Unknown» وهنا يتم التدخل بالتحقق اليدوى من جانب المستفيد أو المطور لاستكمال الاختبار وتحديد النتيجة بدقة True/False. يحتوى العمود الثالث فى الوثيقة على وصف لخطوة الاختبار أو نقطة التحقق وتأخذ نقطة التحقق صيغة السؤال دائماً. تهتم وثيقة الاختبار الحالية بما يلى:

- اختبار عملية الدخول على النظام بنجاح (4V، 6V) أو الفشل فيها (2V).
- عرض الرسائل المرسلة للموظف المتصل (8V، 10V، 12V).

- إرسال رسالة معينة (14V).
- عرض آخر رسالة تم إرسالها (19V).
- إعادة قاعدة بيانات الرسائل إلى الوضع السابق (16V).
- التحقق من الخروج من النظام بنجاح (21V).

الوصف	النتيجة	الخطوة/نقطة التحقق
أدخل اسم المستخدم وكلمة المرور "login" غير صحيحة هل نجحت عملية الدخول ؟	False	1S 2V
أدخل اسم المستخدم وكلمة المرور "login" صحيحة هل نجحت عملية الدخول ؟	True	3S 4V
تحقق من وجود الموظف "get employee" هل الموظف موجود ؟	True	5S 6V
استعرض عدد الرسائل المرسلة "count of messages" هل تم عرض العدد ؟	True	7S 8V
استعرض قائمة الرسائل المرسلة "list of messages" هل تم عرض القائمة ؟	True	9S 10V
استعرض بيانات رسالة مرسلة "get outmessage" هل تم عرض بيانات الرسالة ؟	True	11S 12V
أرسل الرسالة السابقة لنفسك "email outmessage" هل تم وصول الرسالة على عنوانك ؟	Unknown	13S 14V
استعرض بيانات الرسالة المرسلة في الخطوة السابقة "get outmessage" هل تم عرض بيانات الرسالة ؟	False	15S 16V
استرجع قاعدة البيانات للوضع السابق لها قبل إجراء الاختبار استعرض بيانات الرسالة المرسلة في الخطوة رقم 14 "get outmessage" هل تم عرض بيانات الرسالة ؟	True	17S 18S 19V
أغلق التطبيق EM هل تم الخروج بنجاح ؟	True	20S 21V
اخرج أو أعد تنفيذ الاختبار		22S

شكل (٩-١٢) وثيقة سيناريو اختبار القبول للتطبيق EM

٩-٣ اختبار البرمجيات Software Testing:

تبدأ عملية الاختبار للبرامج بعد بداية عملية البرمجة بفترة قصيرة وتستمر معها حتى نهاية اختبار جميع البرامج المنتجة. فبعد الانتهاء من برمجة أى وحدة برمجية Module يتم اختبارها بمفردها أولاً، ثم اختبارها باعتبارها جزءاً من مهمة أو برنامجاً أكبر، ثم أخيراً اختبارها على أنها جزء من النظام العام. ومع أن عملية الاختبار تتم فى أثناء عملية التنفيذ، ولكن فى الحقيقة يتم التخطيط والتجهيز لها فى مراحل سابقة، وذلك بتحديد الوظائف المطلوب اختبارها وإعداد بيانات الاختبار لها. تبدأ أولاً عملية التخطيط لاختبار البرامج فى مرحلة التحليل باختيار الوظائف وإعداد بيانات الاختبار، ثم فى مرحلة التصميم يتم إعداد خطة الاختبار للوحدات البرمجية وخطة اختبار التكامل بين الوحدات، ثم خطة الاختبار العامة للنظام. وفى مرحلة التنفيذ يتم تطبيق تلك الخطط فعلياً لاختبار النظام. يعتبر الغرض الحقيقى من تلك الخطة الثلاثية لاختبار النظام هو تحسين عملية الاتصال بين الأفراد القائمين على تطوير المشروع وتحديد دور كل فرد منهم فى عملية الاختبار. وعلى هذا فإنه يمكننا اعتبار أن عملية الاختبار هى بمنزلة مشروع داخل المشروع الحالى: لأنها تحتاج إلى خطة ومراحل لتنفيذها ومتابعتها. يمكننا تصنيف أنواع الاختبارات المختلفة فيما يلى: (Frank Tsui, 2004).

نوع الاختبار	يدوى	ألى
ساكن Static	فحص Inspection	تدقيق لغوى Syntax Checking
متحرك Dynamic	مراجعات تنابعة walkthroughs تدقيق مكتبى Desk checking	اختبار الوحدة Unit Test اختبار التكامل Integration Test اختبار النظام System Test

يمكن أن تكون عملية الاختبار ساكنة Static أى تكون قبل تنفيذ البرامج، أو تكون Dynamic أى أنها تكون وقت التنفيذ للبرامج. ويمكن كذلك تصنيف عملية الاختبار فهى يمكن أن تتم يدوياً Manual أو آلياً باستخدام الحاسب الآلى. فيما يلى عرض مختصر لأنواع الاختبار.

الفحص Inspection:

هو أسلوب اختبار يدوى ساكن Static يقوم فيه الشخص المختص بفحص البرنامج واصطياد الأخطاء اللغوية المعروفة Syntax errors. تؤدى هذه الطريقة إلى اكتشاف نحو (٦٠٪) أو أكثر من الأخطاء الموجودة فى البرنامج.

المراجعة التتابعية Walkthrough،

هى أسلوب اختبار يدوى ساكن يتم فيه توثيق ومراجعة أجزاء برمجية صغيرة بالاشتراك مع المبرمج المنتج لها وذلك بطريقة دورية لتجنب أغلب الأخطاء أولاً بأول وقبل عملية اختبار البرنامج بأكمله. بالطبع تؤدي هذه الطريقة إلى تقليل كمية الأخطاء المتوقعة فى أثناء اختبار البرامج الكبيرة الحجم.

تدقيق مكتبى Desk checking،

هى طريقة يدوية يتم فيها مراجعة البرنامج مكتبياً باستخدام الورقة والقلم، حيث يقوم المراجع بتنفيذ كل خطوة من البرنامج يدوياً باستخدام حالات اختبار محددة Test Cases. وهنا يقوم المراجع بدور الحاسب، فيقوم بتنفيذ كل خطوة ومقارنة النتائج.

تدقيق لغوى Syntax Checking،

هو أسلوب اختبار آلى ولكن ساكن، حيث يتم تدقيق البرنامج لغوياً فقط باستخدام مترجمات لغات البرمجة Compilers. ويتم ذلك قبل تنفيذ البرنامج للتأكد من خلوه من الأخطاء اللغوية ولكن بالقطع لا يمكن اكتشاف الأخطاء المنطقية بالبرنامج.

اختبار الوحدة Unit Test،

هو أسلوب اختبار آلى يتم فيه تنفيذ الوحدات البرمجية الفرعية Modules منفصلة. ويتم اكتشاف الأخطاء الموجودة بكل وحدة برمجية على حدة.

اختبار التكامل Integration Test،

هو أسلوب اختبار آلى يتم فيه تنفيذ واختبار تكامل الوحدات البرمجية معاً بالتدرج بأسلوب Top-Down. فتبدأ بتنفيذ واختبار الوحدة البرمجية الأساسية Root مع إحدى الوحدات البرمجية الفرعية. وبعد ذلك يتم إضافة وحدة برمجية فرعية أخرى واختبارها مع الوحدات السابقة، وهكذا حتى يتم الانتهاء من اختبار جميع الوحدات البرمجية معاً.

اختبار النظام System Test،

هو أسلوب اختبار آلى مشابه تماماً للأسلوب السابق ولكن مع الفارق أنه فى هذه الحالة يتم تجميع برامج Programs تؤدي وظيفة متكاملة بعضها مع بعض لاختبارها، بدلاً من تجميع وحدات فرعية صغيرة Modules بعضها مع بعض كما فى الطريقة السابقة. ويتم اختبار تكامل البرامج بعضها مع بعض أيضاً بالتدريج، كما سبق، حتى يتم اختبار النظام العام كله System.

اختبار القبول Acceptance Test،

بعد الانتهاء من عملية اختبار البرامج يأتي دور اختبار القبول Acceptance Test وهو يتم عادة من قبل المستخدم User. وهذا الاختبار للنظام يأتي على مرحلتين أساسيتين هما:

١- اختبار أولى Alpha Test،

يتم فيه اختبار النظام باستخدام بيانات مصممة تحاكي بيانات النظام Simulated data. يتم إجراء عدة اختبارات للنظام فى هذه المرحلة للتأكد من تحقق معايير الصحة والأمن الأساسية لأى نظام مثل:

اختبار الاسترجاع Recovery Test،

يتم فيه إجبار النظام على الانهيار للتحقق من إمكانية استرجاع النظام لبياناته بأمان وبدون فقد للمعلومات.

اختبار الأمن Security Test،

يتم فيه اختبار آلية حماية النظام من الاختراق.

اختبار التحميل Stress Test،

يتم فيه تحميل النظام بعدد كبير من العمليات من عدة مستخدمين فى وقت واحد أو إدخال بيانات غير مكتملة، ثم تحليل النتائج لقياس مدى تحمل النظام للتحميل الزائد عند التشغيل.

اختبار الأداء Performance Test:

يتم فيه تحديد كيفية أداء النظام في بيئات عمل مختلفة. فمثلاً يتم اختبار النظام من خلال أنظمة تشغيل مختلفة ومواصفات شبكات مختلفة وأجهزة مختلفة.

٢- اختبار تمهيدى Beta Test:

يتم في هذه المرحلة تشغيل النظام عند المستخدم في بيئة العمل الطبيعية، باستخدام بيانات طبيعية لاختبار مدى ملائمة البرامج والتوثيق والتدريب المقدم لمتطلبات المستخدم. ويتم أيضاً في هذه المرحلة اكتشاف الأخطاء التي لم تظهر في مراحل الاختبار السابقة لكونها تتعامل مع بيانات طبيعية لم تكن في الحسبان عند تجهيز بيانات الاختبار في المراحل السابقة.

الفصل العاشر

تصميم واجهات المستخدم

User Interface Design

بينما يهتم التصميم المعماري للنظام بالهيكل الداخلى والأعمال التفصيلية داخل نظام الأعمال، فإن تصميم واجهات المستخدم يهتم بالمظهر الخارجى للنظام وهو يؤدي إلى جذب العميل وكسب رضاه. فبالإضافة إلى المجهود المبذول في تحقيق أهداف نظام الأعمال بدقة وكفاءة، فإنه لابد من بذل مجهود آخر لا يقل عنه أهمية لتصميم واجهة استخدام النظام UI تتسم بالسهولة والجاذبية وتساعد المستخدم على تنفيذ مهامه بشكل أسرع وأبسط من الطرق التقليدية التي كان يستخدمها من قبل. تتسم معظم تطبيقات الأعمال باعتمادها أسلوب الخادم والعميل Client/Server الذي يتكون من عدة طبقات مثل: طبقة خادم التطبيقات Application Server وطبقة خادم الويب Web Server، وطبقة خادم قواعد البيانات Database Server وطبقة العميل Client. يمكننا تصنيف طبقة العميل إلى صنفين:

عميل مبرمج Programmable Client،

هو العميل الذي يمتلك نسخة من التطبيق والبرامج المنفذة له ويمكن أيضاً أن يمتلك قواعد البيانات التي يتطلبها التطبيق، أى أنه يتم نشر جميع محتويات التطبيق محلياً عند العميل. وأحياناً لا يتم نشر قواعد البيانات عند العميل ولكن يتم تحميل البيانات من خادم قواعد البيانات المتصل به إلى ذاكرة التطبيق ويتم معالجتها محلياً ثم إعادة تسجيلها في قواعد البيانات مرة أخرى عند الحاجة لذلك.

عميل متصفح Browser Client،

هو عميل يعتمد دائماً على خادم الويب في تحميل البيانات المطلوب عرضها على صفحة العميل، وكذلك إرسال البيانات المدخلة من خلال صفحة العميل إلى الخادم لمعالجتها وتخزين البيانات الضرورية في قاعدة البيانات واسترجاع النتائج إلى العميل عند الضرورة. أى أن العميل المتصفح لا يقوم بأى عمليات معالجة باستثناء بعض عمليات التدقيق البسيطة التي يجريها على البيانات المدخلة للتأكد من اكتمالها ومطابقتها للمواصفات المطلوبة قبل إرسالها للمعالجة عند الخادم. تأخذ البيانات

المقدمة عند العمل شكل صفحات HTML، وسوف نقدم الآن عرضاً لبعض طرق تصميم وبرمجة واجهات المستخدم باستخدام لغة جافا.

١٠-١ إرشادات لتصميم واجهة المستخدم User Interface Design Guidelines:

تركز عملية تصميم واجهات المستخدم للنظام على صفة "استخدام النظام Usability" التي تمثل الشكل الخارجى للنظام. يتسم التصميم الجيد لواجهات المستخدم بالخصائص التالية:

- سهولة الاستخدام.
- سهولة التعلم.
- زيادة كفاءة المستخدم.
- سرعة إنجاز المهام.
- تقليل الأخطاء فى أثناء الاستخدام.

هناك مجموعة من الإرشادات التي تساعد على تصميم واجهات المستخدم بشكل جيد يحقق الهدف منها. يمكننا تقديم هذه الإرشادات من خلال النقاط الأربع التالية:

١- تحكم المستخدم:

من الإرشادات المهمة جداً فى أثناء تصميم واجهات المستخدم هو إعطاء المستخدم ميزات التحكم فى تشغيل النظام وتوجيهه من خلال مجموعة التفاعلات بين المستخدم والنظام. وبالفعل نجد أن واجهات المستخدم الحديثة خصوصاً التي تعمل فى بيئة الإنترنت Web Based توفر للمستخدم ميزات التحكم فى البرنامج وتحديد الوجهة التي يرغب فيها لتنفيذ المهام وليس العكس كما كان الحال فى التطبيقات القديمة. أما الآن فباستخدام التقنيات الحديثة أصبح التحكم فى يد المستخدم عند تشغيل التطبيق وأصبح هو الذى يحدد للبرنامج ما هو المطلوب تنفيذه فى الخطوة التالية، ويمكن أيضاً أن يوقف عملية التنفيذ قبل انتهائها. ومن المهم أن تكون واجهة المستخدم سهلة الاستخدام وتستخدم مصطلحات وتعبيرات نظام الأعمال الحقيقية وليست مصطلحات الحاسب الآلى. ويجب أن يراعى مصمم واجهة المستخدم عملية

إعلام أو تنبيه المستخدم بوضوح عن بعض التغييرات التي يمكن أن تحدث في بعض مكونات واجهة المستخدم مثل: (عناصر القوائم Menu Items أو مفاتيح التحكم Command Buttons) نتيجة لحالة التطبيق الحالية في أثناء تنفيذ المهام. ومن الضروري أيضاً أن يراعى عند تصميم واجهة المستخدم إظهار معلومات للمستخدم عند قيام البرنامج بتنفيذ بعض المهام التي لا يجب توقيفها قبل استكمالها لضمان سلامة البيانات ومن ثم كفاءة النظام.

٢- اتساق الواجهات:

يجب أن تتوافق واجهات المستخدم مع المعايير المستخدمة داخل المؤسسة من ناحية الأسماء المستخدمة لمكونات الواجهة والاختصارات المسموحة والأماكن المناسبة لوضع المحتويات في الواجهة، وكذلك تحديد عناصر التحكم المناسبة مثل: قوائم التحكم menus ومفاتيح التحكم buttons وبعض مفاتيح الوظائف في لوحة الإدخال keyboard functions. يجب التأكد من تحقق عملية التوافق هذه في جميع واجهات التطبيق الخاصة بالمؤسسة مما يضمن اتساق واجهات النظام العام. يجب على مصمم واجهات التطبيق تجنب استخدام اتجاهات غريبة وغير مألوفة للمستخدم بفرض الإبداع أو الجاذبية من وجهة نظره، ولكنها يمكن أن تعوق عملية استخدام التطبيق مما يؤثر سلباً في كفاءة استخدام النظام. ومع أن إرشادات تصميم الواجهات تدعم فكرة استخدام الأساليب التي اعتاد عليها المستخدم عند التشغيل، ولكن من الأفضل الاستفادة من الأساليب الحديثة والتسهيلات التي تقدمها التقنيات الحديثة التي تعزز من جودة وكفاءة استخدام النظام. تتميز أيضاً الواجهات جيدة التصميم باتساق الألوان وحسن استخدامها واستخدام المسافات المناسبة بين عناصر الواجهة وتجميع العناصر ذات العلاقة في مكان واحد أو بمعنى آخر تصنيف محتويات الواجهة في شكل مجموعات مما يسهل من عملية استخدامها.

٣- واجهات متسامحة:

يجب أن تتسم واجهات التطبيق بالتسامح نحو تصرفات المستخدم غير الطبيعية مثل: إدخال بيانات غير صحيحة أو محاولة تنفيذ مهام غير مناسبة للحالة الحالية للتطبيق. وبمعنى آخر فإن الواجهة المتسامحة هي التي تشجع المستخدم على استكشاف الخيارات المختلفة للتطبيق من دون التأثير في استمرار تشغيل النظام.

بنجاح. يتم تنفيذ ذلك عن طريق استخدام رسائل توجيهية للمستخدم عند حدوث أحد الاستثناءات Exceptions في أثناء التشغيل. تختلف طبيعة رد فعل التطبيق نحو الأفعال غير المتوقعة من ناحية المستخدم. فأحياناً يكون رد الفعل في شكل رسالة تنبيهية، وأحياناً يكون من خلال عدم السماح أساساً بحدوث هذا الفعل بعدم تمكين المستخدم من إجرائه. وفي جميع الأحوال يجب أن يتميز التطبيق بالصمود أمام هذه الأفعال غير المتوقعة أى أنها لا يجب أن تؤدي إلى سقوط النظام. ولكن يجب أن يتم إعلام المستخدم بطبيعة المشكلة الحالية نتيجة استخدامه للتطبيق ولماذا لا يستطيع التطبيق تنفيذ الأفعال المطلوبة، ويفضل أيضاً أن يخبره بكيفية تصحيح الفعل أو عرض الأفعال البديلة والمسموح بها في التطبيق.

٤- تكيف الواجهة:

من مواصفات التصميم الجيد لواجهة المستخدم أن تكون متوافقة مع أكبر عدد من المستخدمين. فمن الممكن أن يكون هناك اختلاف في إمكانيات المستخدمين للنظام من حيث الخبرة في استخدام الواجهات والتفاعل معها أو من ناحية الوظائف التي يمكنه أن يستخدمها أو المقدرة الشخصية للأشخاص الذين لديهم إعاقة أو قدرات خاصة. وهنا يجب أن يتم الأخذ في الاعتبار عند تصميم واجهات النظام أن تكون متكيفة مع إمكانيات المستخدم، أى أنها تسمح للمستخدم بتعديل مواصفات الواجهة حسب قدراته وصلاحياته في استخدام النظام. ومن الممكن أيضاً أن يتم تنفيذ النظام من خلال أنظمة تشغيل مختلفة، وهنا يجب أن يتكيف التطبيق مع نظام التشغيل الحالي. ويتضمن التكيف أيضاً تغيير لغة التطبيق لتلائم لغة المستخدم أو تغيير الخطوط وحجمها أو تغيير عملة المستخدم أو التغيير في الوقت المحلي وشكل التاريخ أو شكل أرقام الهواتف وغيرها من الخصائص التي يمكن تغييرها في واجهات التطبيق لتلائم طبيعة المستخدم. ومن الأشياء المهمة أيضاً في حالة المستخدمين ذوي القدرات الخاصة أن تتكيف واجهة المستخدم لتلائم قدراتهم. ففي حالة المستخدم الضعيف يجب أن تتضمن الواجهة إمكانيات التعامل مع الأصوات أو استخدام طريقة برايل. وفي حالة المستخدم الأصم يجب أن تعتمد الواجهة على الإمكانيات المرئية بشكل موسع وتعويض أى معلومات صوتية.

١٠-٢ مكونات واجهة المستخدم User Interface Components

تعتمد عملية برمجة واجهات التطبيق على الاستفادة من مكتبة الأصناف Class Library الجاهزة والمتوفرة بشكل كبير من خلال عدد من شركات البرمجيات. تحتوى مكتبة الأصناف على عدد كبير من المكونات التى تستخدم فى بناء واجهات المستخدم بأنواعها المختلفة. فمثلاً تحتوى مكتبة الأصناف فى لغة جافا على حزمة خاصة بمكونات واجهات المستخدم تسمى "Swing" وحزمة أخرى تسمى "AWT". يمكننا أداة Swing من تطوير تطبيقات تتسم بإمكانية تشغيلها على عدة أنظمة تشغيل وإمكانية تخصيص واجهات المستخدم حسب طبيعة التطبيق والمستخدم. تحتوى أداة Swing على مجموعة من المكونات التى تبدأ دائماً بالحرف J مثل: JMenuBar, JButton, JDialog, وغيرها التى يتم استخدامها فى أثناء تصميم واجهات التطبيق. يمكننا تصنيف هذه المكونات فى شكل مجموعات على النحو التالى:

- الحاويات Containers.

- القوائم Menus.

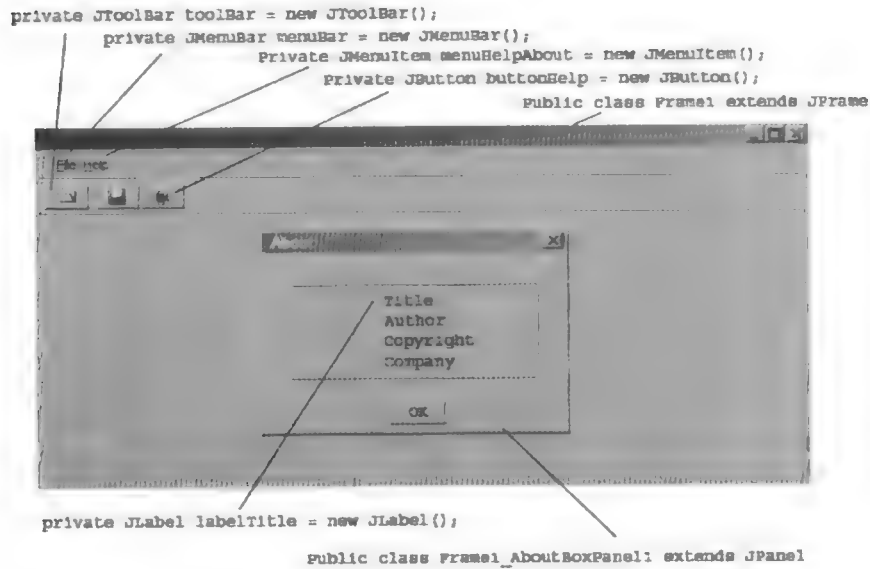
- أدوات التحكم Controls.

وسوف نقوم بعرضها بإيجاز فى الفقرات التالية.

١٠-٢-١ الحاويات Containers

تمثل الحاويات مجموعة الكائنات التى توضع على واجهة سطح المكتب ويمكن أن تحتوى على مكونات أخرى مثل القوائم وأدوات التحكم وحتى يمكن أن تحتوى على حاويات أخرى. تمثل المكونات Windows, Dialogs, Panes, Panels بعض الأمثلة على الحاويات المستخدمة فى تصميم واجهات المستخدم بلغة جافا. تحتوى حزمة Swing على مجموعة من الأصناف التى تستخدم لإنتاج كائنات الحاويات. بعض هذه الكائنات يستخدم فى المستوى الأول للواجهة مثل: JWindow, JFrame, JDialog, JTable, والبعض الآخر يستخدم فى المستويات التالية للواجهة مثل: JApplet, JPanel, JOptionPane, JScrollPane وغيرها. تحتوى الواجهة عادة على نافذة رئيسية واحدة ومجموعة من النوافذ الفرعية. ويستخدم المكون JPanel لتطبيق إحدى النوافذ الفرعية التى تقوم بعرض رسائل للمستخدم أو لإدخال بيانات مثل Dialog Box. يوضح شكل (١٠-١) مثلاً على استخدام نافذة فرعية داخل نافذة رئيسية. تعتبر

النافذة الرئيسية من المكون JFrame والنافذة الفرعية من المكون JPanel. تحتوى النافذة الرئيسية على مكون القائمة JMenuBar ومكون شريط الأدوات JToolBar وبعض مفاتيح التحكم فى النافذة مثل: التصغير والتكبير والإغلاق. ويحتوى مكون القائمة على مكونات أخرى فرعية مثل JMenuItem. وكذلك يحتوى مكون شريط الأدوات على مجموعة من مفاتيح التحكم JButtons. تتميز النافذة الرئيسية بإمكانية تغيير حجمها من أى جنب أو من الزوايا ويمكن أيضاً تحريكها من شريط العنوان.



شكل (١٠-١) استخدام النوافذ فى واجهات المستخدم

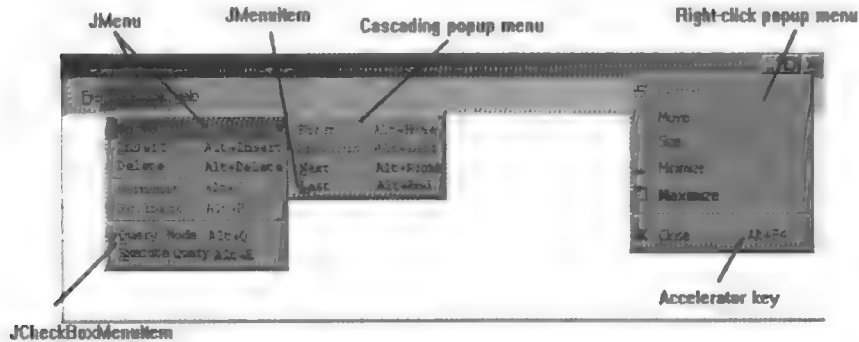
. يوجد مكون آخر يسمى Applet يستخدم فى تنفيذ برامج داخل المتصفح، ويتم تحميل تلك البرامج من خادم الويب إلى صفحة المتصفح. تقدم أداة Swing أسلوباً لتنظيم وضع المكونات داخل الواجهة يسمى "LayoutManager" الذى يتضمن مكونات مثل BorderLayout, FlowLayout, GridLayout, BoxLayout. يعتبر أسلوب BorderLayout هو الأسلوب الافتراضى لتنظيم وضع المكونات داخل حاويات واجهة التطبيق. تقدم أداة Swing أيضاً صنفاً خاصاً يدعم وضع النوافذ فى طبقات Layers ويسمى JLayerPane. يحتوى الصنف على طرق خاصة بعمليات تحريك المكونات من الأمام إلى الخلف والعكس أو وضعها فى أماكن معينة. يتم وضع المكونات

فى الطبقة الافتراضية Default Layer بشكل متداخل Overlap ويتم تحديد النافذة الأمامية للواجهة حسب اختيار المستخدم. ويسمح الصنف JLayerPane بتعريف طبقات بمواصفات خاصة مثل:

- طبقة عائمة Palette Layer: تظهر فوق الطبقة الافتراضية.
- طبقة إجبارية Modal Layer: تظهر على رأس كل النوافذ النشطة مثل شريط الأدوات.
- طبقة منبثقة Popup Layer: تظهر مؤقتاً فوق طبقات أخرى.
- طبقة سحب Drag Layer: تظهر المكون المطلوب سحبه قبل إلقائه فى طبقة أخرى.

٢-٢-١٠ القوائم Menus:

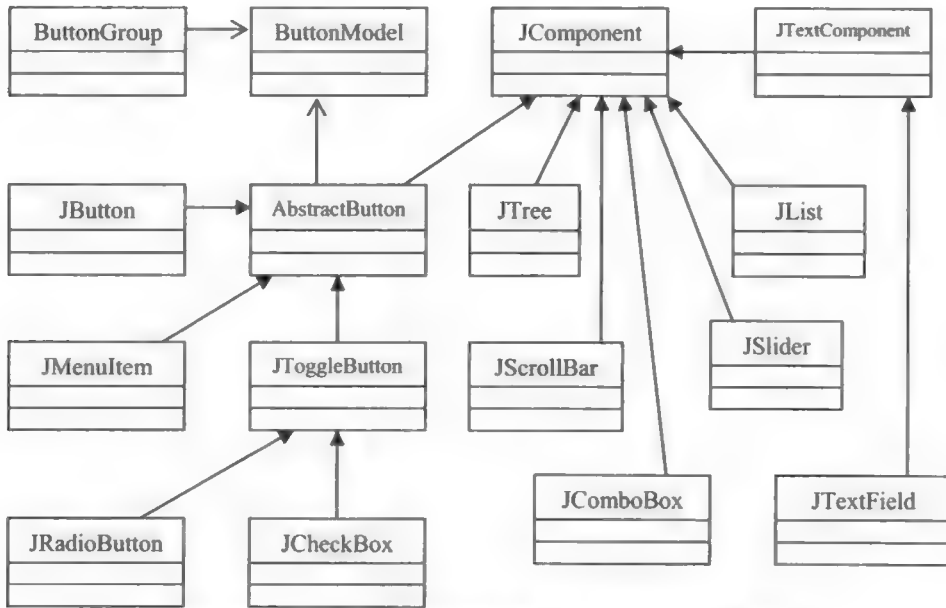
تحتوى معظم نوافذ واجهات التطبيق على مكون القائمة الذى يتضمن مجموعة من عناصر التحكم فى تشغيل التطبيق. تقدم أداة Swing مجموعة من الأصناف الخاصة بالقوائم. يوضح شكل (٢-١٠) مثلاً لواجهة تطبيق تحتوى على شريط القائمة JMenuBar الذى يتضمن كائنات JMenuItem. ويحتوى كل كائن JMenuItem على مجموعة من عناصر القائمة JMenuItem مثل Insert أو GoTo الذى يحتوى على عناصر أخرى فرعية فى شكل قائمة منبثقة JPopupMenu. يمكن أن يأخذ عنصر القائمة JMenuItem شكل صندوق الاختيار JCheckBoxMenuItem أو شكل مفتاح الاختيار JRadioButtonMenuItem. ويمكن أيضاً تشغيل عناصر القائمة باستخدام مفاتيح السرعة Accelerator Keys.



شكل (٢-١٠) استخدام القوائم فى واجهة التطبيق

٣-٢-١٠ أدوات التحكم Controls:

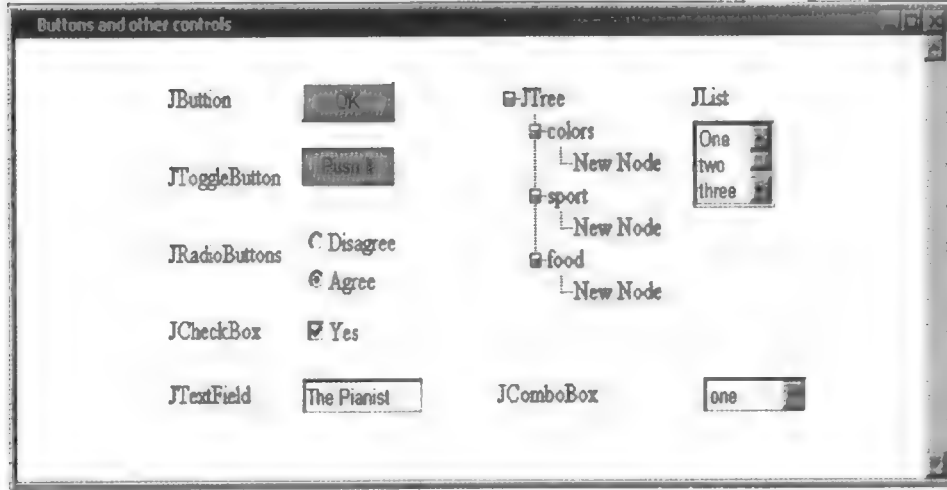
تعتبر أدوات التحكم بمنزلة القلب لنموذج أداة Swing حيث إنها تقوم بعمليات التقاط البيانات والأفعال التي يطلبها المستخدم وتفسيرها وتطبيقها من خلال واجهة التطبيق. يوضح شكل (٣-١٠) نموذج الأصناف لأدوات التحكم في أداة Swing. يوضح الشكل وجود أربعة أنواع من مفاتيح التحكم هي: JButton, JRadioButton, JCheckBox, JMenuItem، وهي ترث من الصنف المجرد AbstractButton. بالإضافة إلى مجموعة أخرى متنوعة من أدوات التحكم مثل: JTree, JList, JComponent إلى مجموعة أخرى متنوعة من أدوات التحكم مثل: JComponent، وهي ترث من الصنف المجرد JComponent.



شكل (٣-١٠) نموذج الأصناف لأدوات التحكم في أداة Swing

ويوضح شكل (٤-١٠) أشكال أدوات التحكم المذكورة في شكل (٣-١٠). يتم تطبيق أدوات التحكم من خلال مجموعة من الأحداث التي يمكن أن ترافق الأداة. فمثلاً يتضمن كائن التحكم JButton الحدث Click () والحدث DoubleClick () وغيره من الأحداث. تستخدم الأداة JCheckBox للاختيار المتعدد، في حين تستخدم الأداة JRadioButton لاختيار واحد فقط من مجموعة الاختيارات. تستخدم الأداة JComboBox لاستعراض قائمة منسدلة من العناصر واختيار أحدها فقط سواء

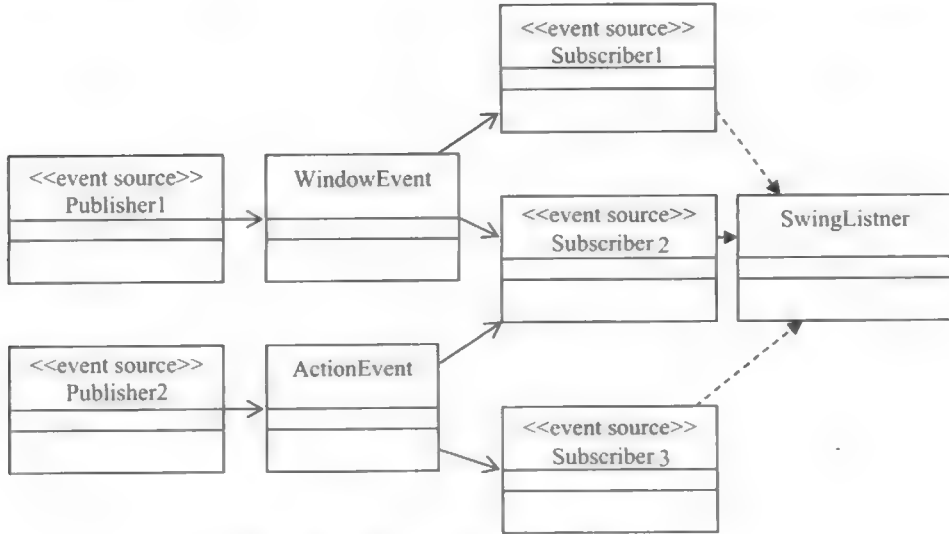
بالنقر على العنصر نفسه أو بكتابة جزء منه في صندوق النص الخاص بالأداة. أما الأداة JList فتستخدم لعرض قائمة عناصر بحجم ثابت Fixed Size ويمكن اختيار مجموعة من العناصر في الوقت نفسه. تستخدم أداة JSlider للتحكم في بعض المدخلات مثل التحكم في الصوت Volume Control. وتستخدم أداة JScrollBar لإمكانية تصفح باقى محتويات أى مكون على الواجهة له حجم أقل من حجم المحتويات له.



شكل (١٠-٤) أدوات التحكم في أداة Swing

١٠-٣ معالجة أحداث واجهات المستخدم User Interface Event Handling؛

تحتاج جميع الأفعال التي يقوم بها المستخدم من خلال واجهة التطبيق إلى عمليات التقاط ومعالجة. فعندما يقوم المستخدم بأداء فعل معين باستخدام وحدات الإدخال المتاحة مثل لوحة المفاتيح أو الفأرة فإنه يتم التقاط هذا الفعل من خلال كائن يسمى كائن مصدر الحدث Event Source Object أو كائن النشر Publisher Object الذي يقوم بعملية الإعلان عن حدوث الفعل للكائن أو الكائنات المقصودة بهذا الفعل ويطلق عليها الكائنات المستمعة Listener Objects أو الكائنات المشاركة Subscriber Objects. يوضح شكل (١٠-٥) مثلاً بسيطاً لمعالجة أحداث واجهة المستخدم. نلاحظ من الشكل أن كل صنف مشارك Subscriber Class يقوم بتطبيق صنف الواجهة Action Listener أو يكون امتداداً لصنف يقوم بتطبيق صنف الواجهة.



شكل (١٠-٥) معالجة أحداث واجهة المستخدم

١٠-٤ الأنماط وواجهات المستخدم Patterns and User Interfaces

لقد أدى ظهور أسلوب واجهات المستخدم الرسومية GUI إلى اعتماد أسلوب البرمجة المبنية على الأحداث Event-Driven Programming بشكل موسع في تطوير التطبيقات. وهنا نجد أن مركز التحكم في البرنامج أو التطبيق أصبح في يد المستخدم، فهو الذي يحدد ما الذي يجب تنفيذه في الخطوة التالية من خلال مجموعة الأحداث الموجودة في التطبيق. كل حدث يتطلب مجموعة من الإجراءات التي تقوم بمعالجته. وقد أدى التوسع في هذا الأسلوب إلى وجود مجموعة من الأنماط التي يمكن استخدامها في تصميم واجهات العديد من التطبيقات. سوف نقدم الآن عرضاً لأربعة أنماط تستخدم على نطاق واسع في تطوير واجهات المستخدم.

١٠-٤-١ نمط المراقب Observer

هناك العديد من مكونات واجهات المستخدم التي تقوم بمراقبة كائنات التطبيق، مما ينعكس على سلوك واجهة المستخدم بناءً على المعلومات التي تم التقاطها من الكائنات التي حدث بها تغيير. يتم استخدام نمط المراقب في واجهات المستخدم للتعرف على التغييرات التي تحدث في مكونات واجهة التطبيق مثل مركب النص Text Box أو

صندوق الاختيار Check Box وغيرها وهي تتطلب تنفيذ إجراء معين يناسب التغيير الذي يحدث مثل استدعاء الطريقة () Text_Changed في حالة حدوث تغيير في محتوى مركب النص.

١٠-٤-٢ نمط النقاش Decorator:

يستخدم نمط النقاش لتجميل شكل واجهة المستخدم وتحسين أسلوب استخدامها لتصبح مناسبة للمستخدم بشكل بسيط ومريح User-Friendly. فمثلاً يفضل دائماً استخدام مكون الحدود Border لتجميل شكل النوافذ والإطارات وبعض مكونات واجهة المستخدم الأخرى. ولذلك فإن نمط النقاش يستخدم عادة لتجميل شكل مكونات واجهة المستخدم. فيمكننا استخدام مركب المنزلق Scroll Bar لتجميل شكل مكون القائمة JList وتحسين أسلوب استخدامها. ويمكننا استخدام مركب شريط القائمة Menu Bar، وشريط الأدوات Tool Bar، والحدود Border لتجميل شكل ظهور مكون الإطار للواجهة Frame.

١٠-٤-٣ نمط سلسلة المسؤولية Chain of Responsibility:

عندما يكون أحد مكونات الواجهة غير قادر على تنفيذ الخدمة المطلوبة الآن، فإنه يقوم بتمريرها إلى المكونات الفرعية له، فيقوم أحد المكونات الفرعية بالاستجابة للطلب وتنفيذه. أى أن المكون الأصلي قد فوض أحد المكونات الفرعية له في تنفيذ الخدمة، ويطلق على هذا الأسلوب اسم "سلسلة المسؤولية" وهو يستخدم على نطاق واسع في واجهات المستخدم خصوصاً في نظم المساعدة Help Systems.

١٠-٤-٤ نمط الأمر Command:

يشبه نمط الأمر نمط سلسلة المسؤولية في المفهوم ولكن يختلف عنه في الأسلوب. يقوم نمط الأمر بتفويض المكون التالى لتنفيذ الخدمة المطلوبة، فإن لم يستطع فينتقل التفويض إلى المكون التالى له حتى يصل إلى المكون الذى يستطيع القيام بتنفيذ الخدمة المطلوبة. أى أن التفويض فى نمط الأمر يكون محدداً لمكون معين داخل الواجهة، وذلك بخلاف نمط سلسلة المسؤولية، إذ يكون التفويض غير محدد ويكون مسموحاً لأى مكون فرعى يمكنه تنفيذ الخدمة. يمكن استخدام هذا الأسلوب فى تنفيذ عمليات التراجع Undo والإعادة Redo فى واجهات المستخدم.

٥-١٠ واجهات المستخدم على الويب Web-Based User Interface؛

من الاتجاهات الحديثة لتطوير تطبيقات النظم هو تمكينها للعمل من خلال بيئة الإنترنت، حيث يمكن استخدامها فى أى وقت فى أى مكان. وبالفعل فقد قامت هذه التقنية الحديثة بجذب انتباه معظم مطورى النظم للاستفادة من المميزات الكبيرة التى تعود على المؤسسات من استخدامها. فبالإضافة إلى إمكانية استخدام التطبيق من أى مكان وعدم التقيد بمواقع محددة داخل أو خارج المؤسسة من خلال شبكة محلية LAN، فإن هذه التقنية تختصر عملية توزيع وتثبيت برمجيات التطبيق على أجهزة العميل وتكتفى فقط باستخدام برنامج متصفح الويب Web Browser الذى يقوم بجلب وعرض معلومات التطبيق الموجودة فى خادم الويب Web Server من خلال شبكة الإنترنت. أى أنه يتم توزيع العمل، بحيث تكون قاعدة البيانات وبرامج التطبيق التى تحتوى على قواعد العمل الخاصة بنظام الأعمال موجودة على خادم التطبيق على الويب Web Server، وتكون البرامج الخاصة بجلب وعرض معلومات التطبيق موجودة على جهاز العميل Client فى شكل واجهات المستخدم GUI. تحتوى واجهة التطبيق على مكونات ساكنة Static باستخدام تعليمات HTML ومكونات متحركة Dynamic باستخدام برمجيات جافا Java Applets وشفيرات جافا Java Script.

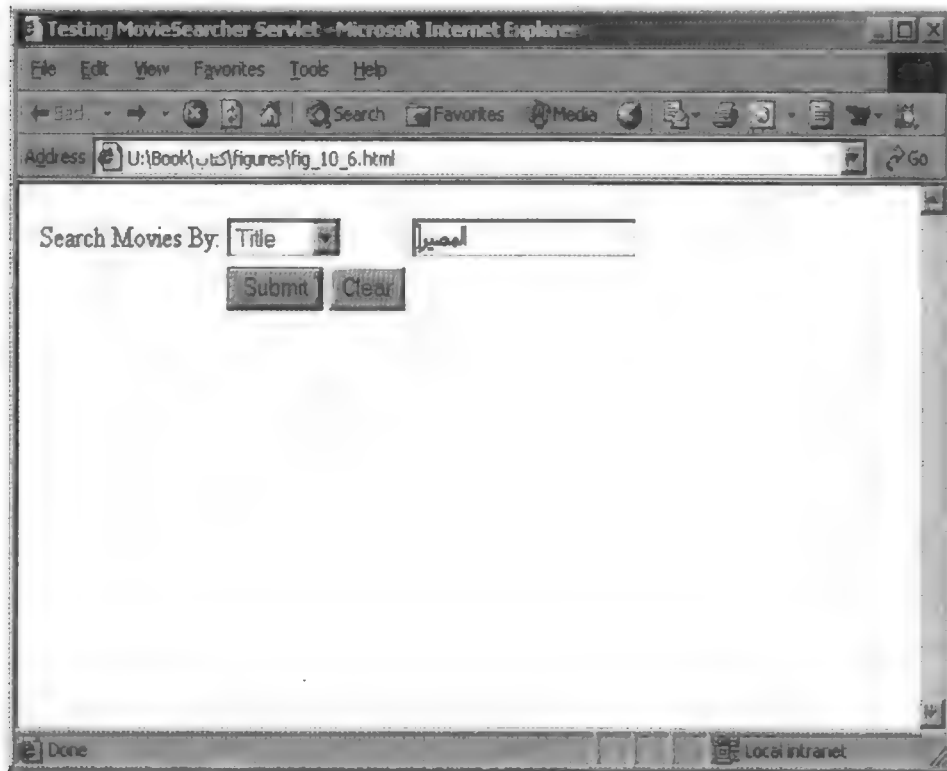
تتسم معظم تطبيقات النظم التى تعمل من خلال شبكة الإنترنت بأنها موزعة على ثلاث طبقات. الطبقة الأولى هى طبقة العميل Web Client، والطبقة الثانية هى طبقة خادم التطبيق Web Server، والطبقة الثالثة هى طبقة خادم قواعد البيانات Database Server.

٦-١٠ تمكين التقنيات لمستخدمى الويب Enabling Technologies for Web Clients؛

تكمُن أهمية أسلوب تنفيذ التطبيقات على الويب فى أنها تلبي رغبات كل من المستخدم والمطور للتطبيقات. فمن ناحية المستخدم فهو لا يهتم كثيراً بمكان تنفيذ التطبيق سواء كان من جهازه أو من جهاز آخر عبر الإنترنت. وبالنسبة لمهندسى النظم فإن عملية فصل واجهة استخدام التطبيق عن برامج التطبيق وقواعد البيانات تعتبر عملية منطقية، بالإضافة إلى تسهيل عملية نشر مكونات التطبيق على كل من الخادم والعميل. سنعرض فى هذا الجزء بعض التقنيات المستخدمة فى تصميم واجهات المستخدم على الويب مثل لغة HTML وبعض لغات إعداد صفحات الويب Scripting Languages وبرمجيات جافا Java Applets.

١٠-٦-١ لغة HTML

تعتبر لغة (HTML) Hyper Text Markup Language هي أكثر اللغات شيوعاً في تصميم صفحات الويب. تأخذ صفحات HTML شكلاً قياسياً بسيطاً يتضمن مجموعة من العلامات Tags التي تسمح بظهور الصفحات بشكل جيد على متصفحات الويب Web Browsers. يوضح شكل (١٠-٦) مثالاً على استخدام لغة HTML لتصميم صفحة واجهة استخدام تقوم بالاستعلام عن الأفلام فى قاعدة بيانات الأفلام Movie Actor. ويتضح من الصفحة أن المستخدم يقوم باختيار أسلوب البحث مثل العنوان Title، ثم يقوم بإدخال اسم الفيلم ليتم تنفيذ البحث وعرض تفاصيل الفيلم. ويوضح شكل (١٠-٧) محتويات ملف HTML الذي يقوم بإنتاج هذه الصفحة الموضحة بشكل (١٠-٦).



شكل (١٠-٦) مثال لصفحة HTML

Sample HTML document

```

<html>
<head>
<!-- comment: مثال لإظهار عنوان لصفحة ويب -->
<title> Testing MovieSearcher Servlet </title>
<body>
  <form method="post" action="/SearchMovie">
    <table border=0 width="70%">
      <tr><td>Search Movies By:</td>
        <td><select name="searchBy">
          <OPTION value="TitleV">Title
          <OPTION value="ActorV">Actor
          <OPTION value="DirectorV">Director
        </select></td>
      <td><input name=searchField type=text></td>
    </tr>
    <tr><td><input type="submit" value="Submit">
      <input type=reset value="Clear"></td></tr>
    </table>
  </form>
</body>
</html>

```

شكل (١٠-٧) محتويات ملف HTML

يبدأ دائماً ملف HTML بالعلامة <html> وينتهي بالعلامة </html>. ويمكن أن يتضمن الملف جزءاً خاصاً في المقدمة <head> يستخدم لإعلام متصفح الويب بمعلومات عن الملف نفسه. ويمكن أيضاً استخدام الجمل التوضيحية Comments في أى مكان في الملف باستخدام العلامات <!--> ، <!-->. يتم إضافة عنوان الصفحة في مقدمة الملف باستخدام العلامة <title> ، </title>. نلاحظ أن معظم علامات لغة HTML تحتاج إلى إغلاق باستخدام الرمز / قبل اسم العلامة. ولكن هناك بعض العلامات التي لا تحتاج إلى إغلاق مثل <option>. ونلاحظ أيضاً من الشكل أن بعض العلامات يكون لها معاملات أو خصائص مثل: علامة <form> التي تتضمن خاصيتين هما (method, action). تستخدم العلامة <body> و </body> لتوضيح الجزء الذي سوف يظهر داخل إطار متصفح الويب عند تنفيذ صفحة HTML. يتم استخدام العلامة <table> لتمثيل البيانات على الصفحة في شكل جدول. ويتم تنظيم البيانات داخل الجدول عن طريق الصفوف <tr> والأعمدة <td>.

يتم في هذا المثال استقبال المدخلات من المستخدم داخل صفحة المتصفح باستخدام العلامة <form> وهي تتضمن طريقة من النوع "post" وهي تعنى أنه سيتم إرسال البيانات المجمعة داخل نموذج الصفحة إلى الخادم أو مصدر خارجي للمعلومات مثل قواعد البيانات. يتم تحديد عنوان الخادم Server URL من خلال المعامل action للعلامة <form>. يقوم المستخدم باختيار أحد بيانات القائمة باستخدام العلامة <select> وإدخال بيانات في مركب النص text باستخدام العلامة <input>. هناك نوع آخر من الإدخال وهو <submit> هو يمثل بمفتاح إدخال button يقوم بإحالة البيانات الموجودة بالنموذج <form> إلى عنوان الخادم. وهناك أيضاً النوع <reset> الذى يستخدم عادة لإعادة محتويات النموذج إلى حالتها الأصلية (حقول فارغة أو قوائم ذات اختيار مبدئى أو افتراضى). بالطبع تحتوى لغة HTML على مجموعة كبيرة من العلامات التى تستخدم لتصميم صفحات الويب ولكن لا يتسع المجال هنا لعرضها جميعاً، ولكن نكتفى بعرض بعض العلامات التى نستخدمها فى هذا المثال التوضيحي.

١٠-٦-٢ لغات إعداد صفحات الويب Scripting Languages

تستخدم لغة HTML بعض الأدوات البرمجية لدعم عمليات التعامل مع المستخدم من خلال متصفح الويب. تعتبر لغات VB Script, Java Script من تلك الأدوات المستخدمة على نطاق واسع فى ملفات HTML لإجراء عمليات التحقق السريعة للمدخلات على الصفحة ولتشغيل الصور المتحركة Animation والعمليات الأخرى التى تضيف صفة الحركة على صفحة الويب. يجب تعريف نوع اللغة المستخدمة فى مقدمة ملف HTML. يوضح شكل (١٠-٨) مثالا لاستخدام لغة Java Script للتحقق من صحة البيانات المدخلة عن طريق المستخدم (سطر ٢٢) والتأكد من أنها تقع داخل المدى المسموح (سطر ٢٠). يتم تنفيذ تعليمات التحقق عندما يقوم المستخدم بالضغط على مفتاح Submit، حيث يتم استدعاء الدالة () Checkit من خلال المعامل Action المصاحب للعلامة <Form> (سطر ٢٠). تقوم الدالة () Checkit (سطر ٥) بالتحقق من القيمة المدخلة (سطر ٢٢) للتأكد من وقوعها ضمن المدى المعرف بالقيمة الصفري Lower والقيمة الكبرى Upper. فإذا كانت القيمة تقع ضمن المدى المحدد تسترجع الدالة القيمة True وغير ذلك تسترجع القيمة False مع إظهار رسالة تشير إلى عدم صحة البيانات المدخلة (سطر ١٢، ١١).

Sample Java Script

```

<HEAD>
  <TITLE> Form Validation </TITLE>
  <SCRIPT LANGUAGE="JavaScript">
    <!--
      function checkIt(int lower, int upper) {
        var strval = document.sampleForm.percent.value;
        var intval = parseInt(strval);
        if ( lower < intval && intval < upper ) {
          return (true);
        } else {
          alert("Input " + strval + " is out of " + lower +
            "-" + upper + " range");
          return (false);
        }
      }
    <!-- end script -->
  </SCRIPT>
</HEAD>
<FORM NAME="sampleForm" METHOD="post"
  ACTION="/SearchMovie" onSubmit="checkIt(0,100)">
  <P>Percentage given:
  <INPUT TYPE="text" NAME="percent" VALUE="1">
  <BR><INPUT TYPE="submit">
</FORM>

```

شكل (١٠-٨) مثال على استخدام لغة Java Script داخل ملف HTML

١٠-٦-٣ البرمجيات Applets:

البرمج Applet هو برنامج صغير يؤدي وظيفة معينة ويكون مخزناً في خادم الويب ولكن يتم تحميله وتنفيذه من خلال متصفح الويب Web Browser عند العميل. هناك نوعان من البرمجيات: برمج خفيف يسمى Thin Applet، وبرمج ثقيل يسمى Thick Applet. يتفق النوعان في أنهما يحتويان على مجموعة من الأصناف التي يتضمنها التطبيق ويختلفان في حجم الكود الذي يتم تنفيذه من خلال المتصفح. ففي حالة البرمج الخفيف يتم تنفيذ أغلب الوظائف من على الخادم ويتم تحميل جزء بسيط فقط إلى جهاز العميل ليتم تنفيذه من خلال المتصفح. وهو يتميز بالسرعة

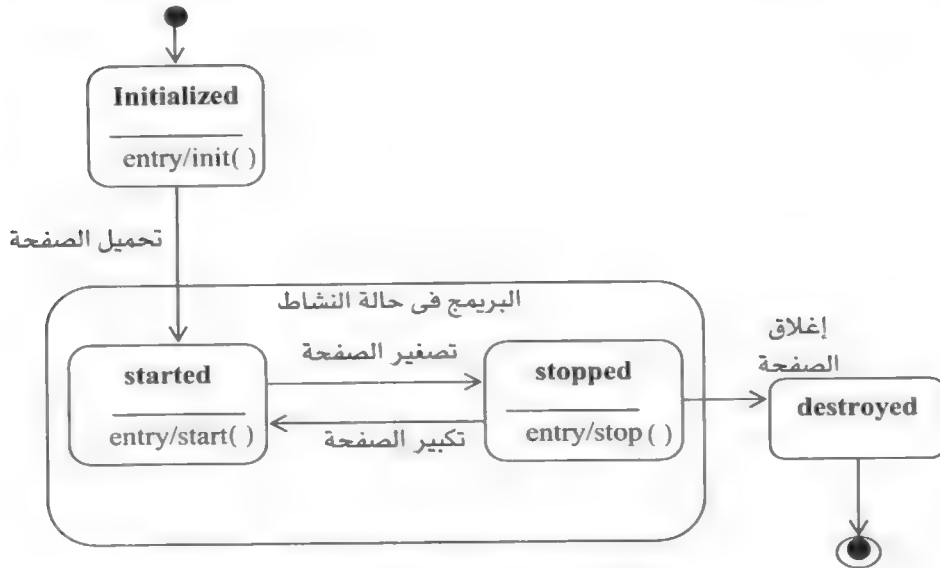
أثناء التنفيذ نظراً لصغر حجمه وهو يناسب الشبكات البطيئة، ولكنه يكون صعباً في الصيانة وتتبع الأخطاء التي تحدث في أثناء التنفيذ نظراً لاعتماده على الخادم بشكل كبير. أما في حالة البريمج الثقيل فإنه يتم تحميل جزء كبير من البرنامج الموجود على الخادم إلى جهاز العميل ويتم تنفيذه من خلال المتصفح، وهو يناسب الشبكات ذات السرعات العالية وتكون صيانتها أسهل نظراً لعدم اعتماده على الخادم بشكل كبير في أثناء التنفيذ. يوضح شكل (١٠-٩) نموذجاً لحالة البريمج حيث إنه يمر بأربع حالات. جدير بالذكر أن عملية إنشاء البريمج Applet Construction تتم قبل استدعائه، وبعد ذلك يقوم المتصفح باستدعاء الطريقة () init لتهيئة الكائن (البريمج) لعملية التحميل. بعد ذلك يدخل البريمج في حالة النشاط وذلك باستدعاء الطريقة start لبدء عملية التنفيذ، ويظل في هذه الحالة طالما كانت صفحة المتصفح مفتوحة. ينتقل البريمج إلى حالة التوقف () stop في حالة أن يكون غير نشط (في أثناء تصغير نافذة المتصفح) وذلك لتوفير الموارد التي يستخدمها البريمج. وعندما يتم إغلاق صفحة الويب ينتقل البريمج إلى حالة الانتهاء () destroy ويتم تحرير الموارد التي كان يستخدمها البريمج وإعادتها إلى حالتها الأولية. يوضح شكل (١٠-١٠) مثلاً على استخدام البريمجات داخل صفحة HTML على متصفح الويب.

يتم استخدام البريمجات في العديد من التطبيقات على الويب مثل: التطبيقات البنكية ومراقبة المخزون وبرامج الاتصال والدردشة عبر الإنترنت وغيرها. يتم عادة تشغيل البريمجات من خلال بيئة مقيدة Restrictive Environment تمنعه من استخدام الموارد الأساسية والحرية للحاسب المضيف Host Computer بحيث لا يكون مسموحاً للبريمج بما يلي:

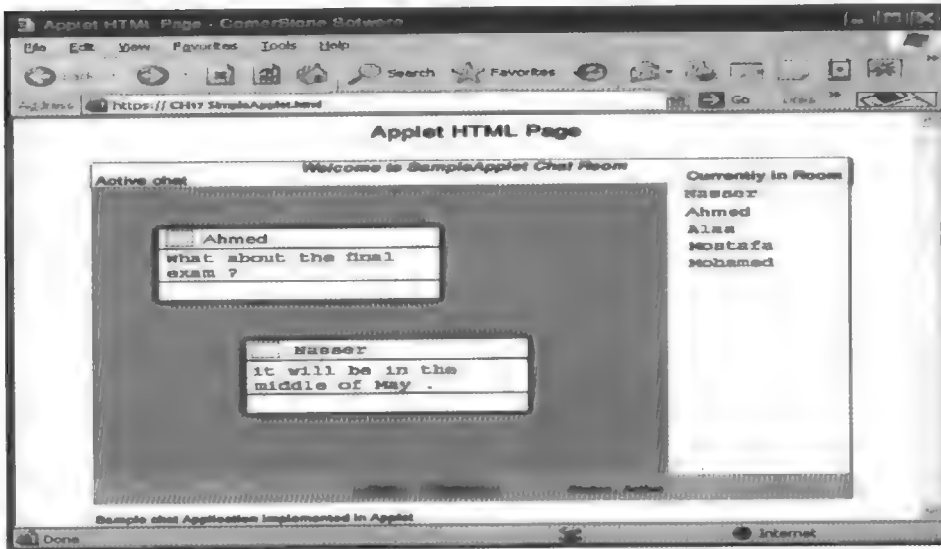
- القراءة من ملفات النظام للجهاز المستضيف.
- كتابة أو حذف ملفات منه.
- تحميل أو تنفيذ برامج أخرى.
- الاتصال عبر الشبكة بأي جهاز آخر غير الخادم القادم منه البريمج.

وتعتبر الوسيلة الوحيدة التي تمكن البريمج من استخدام موارد الجهاز المضيف هي أن يقوم المطور بمنح تلك الصلاحية للبريمج.

بريمج تم تحميله على جهاز العميل



شكل (٩-١٠) نموذج حالة البريمجات



شكل (١٠-١٠) استخدام البريمجات

١٠-٧ تمكين التقنيات ل خادم الويب Enabling Technologies for Web Server

يقوم خادم الويب بإدارة عملية الاتصال بين عملاء الويب والتطبيق الخاص بنظام الأعمال. يتم عادة تطبيق عمليات نظام الأعمال داخل خادم الويب أو من خلال خادم خاص بالتطبيقات Application Server. يقوم خادم الويب أيضاً بالوظائف التالية:

- تقديم المحتوى الديناميكي لصفحات الويب مثل: الصور والرسومات وملفات الصوت، وغيرها.

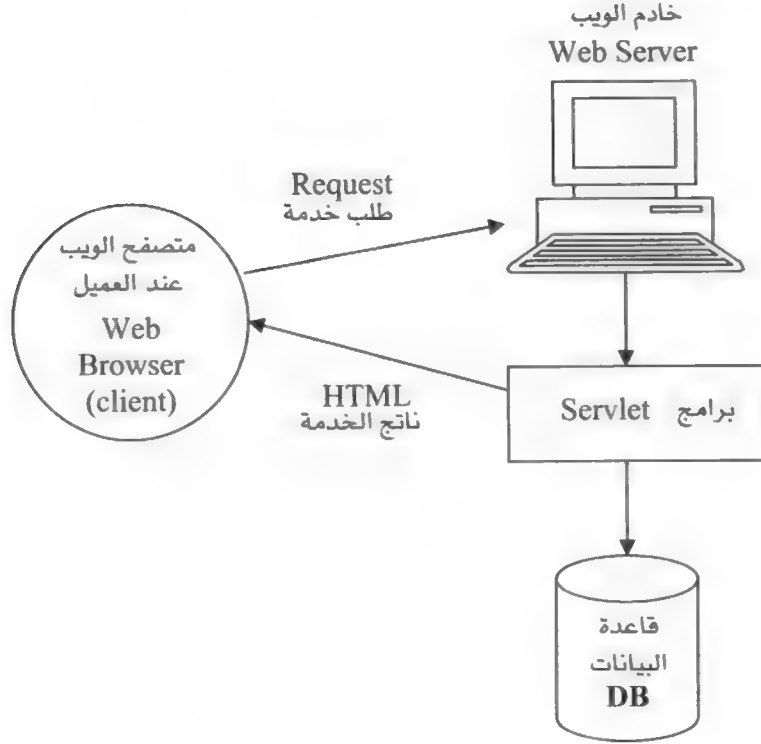
- التحكم في تدفق الشاشات من خلال الروابط في متصفح العميل.
- صيانة الحالات الإجرائية عند العميل.

يتوافر الآن العديد من التقنيات التي تدعم عمليات خادم الويب مثل: Common Gateway Interface (CGI) وهي كانت تستخدم في الفترة السابقة ولكن تم استحداث أسلوبين آخرين أكثر مقدرة منها هما Servlets, JSP عن طريق شركة Sun Micro Systems من خلال تقنية J2EE.

١٠-٧-١ تقنية Servlet

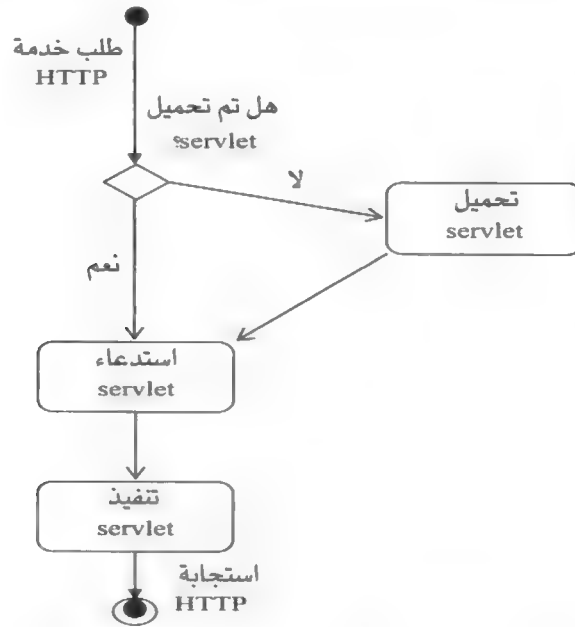
يعتبر ال Servlet هو برنامج جافا يتم تخزينه على خادم الويب Web Server ويقوم بتقديم خدمة معينة للعديد من أجهزة العملاء Clients المتصلة بالخادم عن طريق شبكة الإنترنت. ويتضح الفرق بين Servlet والبريمج Applet في أن Servlet يكون موجوداً دائماً على الخادم ويتم ترجمته وتنفيذه أيضاً من على الخادم. أما البريمج فيتم تحميله إلى جهاز العميل وتنفيذه من خلاله. يوضح شكل (١٠-١١) كيفية الاتصال بين أجهزة العميل وخادم الويب لتنفيذ خدمة معينة باستخدام تقنية Servlet.

يحتوي خادم الويب على برامج Servlets التي تقدم الخدمات المطلوبة للعملاء، ويقوم خادم الويب باستقبال طلب الخدمة من جهاز العميل (يكون غالباً من خلال صفحة HTML أو صفحة JSP) ثم يقوم بالبحث عن Servlet المناسب لتنفيذ الخدمة المطلوبة ويتم تنفيذه من على الخادم واسترجاع النتائج للعميل طالب الخدمة في شكل HTML. يقوم Servlet في أثناء التنفيذ باستخدام موارد الخادم ومن ضمنها قواعد البيانات.

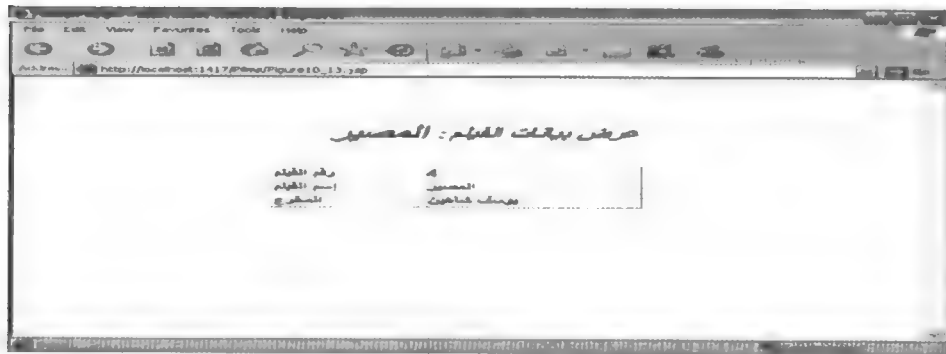


شكل (١٠-١١) الاتصال بين العميل وخادم الويب في تقنية Servlet

يدخل Servlet في حالة النشاط عندما يتم استدعاؤه من الخادم باستخدام الطريقة (`init()`) ويظل في هذه الحالة عند وجود طلبات من العملاء من خلال بروتوكول HTTP. وعندما تنتهي طلبات العملاء يتم استدعاء الطريقة (`destroy()`) لإنهاء Servlet وتحرير الموارد التي كان يستخدمها. يوضح شكل (١٠-١٢) كيفية استدعاء Servlet من خلال طلبات HTTP، حيث يقوم الخادم بالبحث عنه والتأكد من أنه موجود بالذاكرة أم لا، فإذا لم يكن موجوداً يتم تحميله أولاً ثم استدعاؤه من خلال الطريقة (`init()`) ويتم تنفيذه واسترجاع ناتج الخدمة إلى العميل من خلال HTTP Response في شكل HTML. ويوضح شكل (١٠-١٣) مثلاً على استخدام تقنية Servlet لإظهار معلومات مستخرجة من قاعدة بيانات الأفلام الموجودة على الخادم وذلك نتيجة لطلب استدعاء هذه المعلومات من جهاز أحد العملاء.



شكل (١٠-١٢) خطوات استدعاء وتنفيذ Servlet

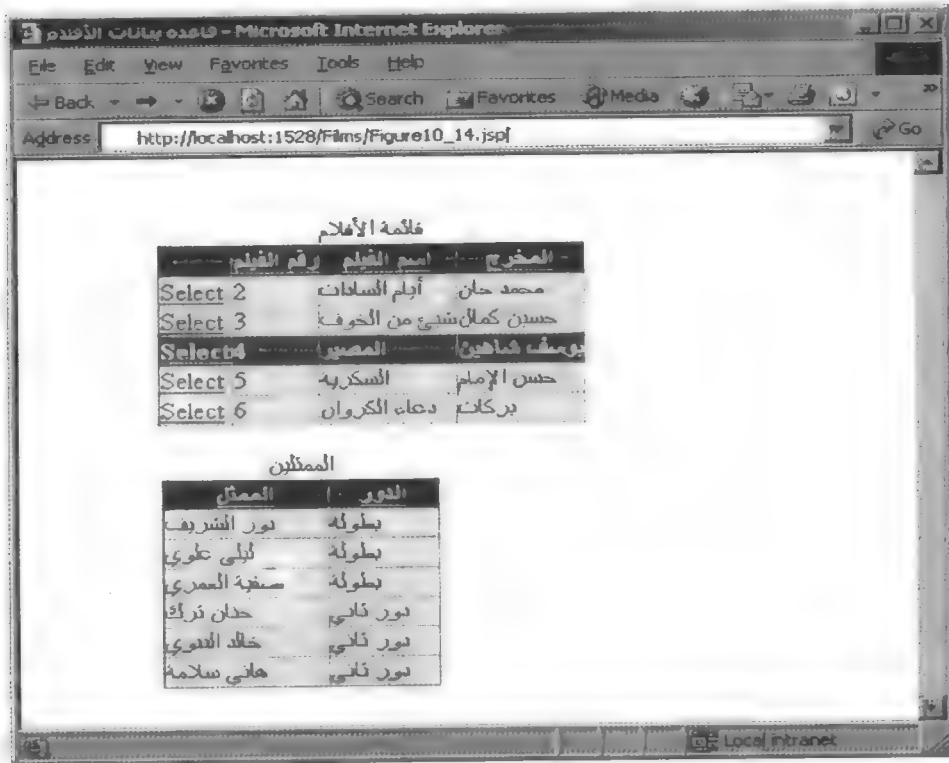


شكل (١٠-١٣) مثال على استخدام ال Servlet

١٠-٧-٢ تقنية JSP

تعتبر تقنية إعداد صفحات خادم الويب بلغة جافا (Java Server Pages (JSP أسلوباً مشابهاً لأسلوب Servlet من حيث الاتصال بين العميل والخادم كما هو موضح

فى الشكل (١٠-١١) ولكنه يختلف عنه من ناحية فصل منطقة العمل Business Logic عن منطقة العرض Presentation: حيث يتم تمثيل منطقة العمل باستخدام Servlet وتمثيل منطقة العرض باستخدام صفحات JSP. ولهذا فإن أسلوب JSP يحتاج إلى خطوتين إضافيتين قبل عملية استدعاء Servlet. يجب تحليل محتويات صفحة JSP أولاً وإنتاج الكود الذى سيتم ترجمته وتجهيزه للاستخدام، وبعد ذلك تتم عملية استدعاء servlet وتنفيذه. أى أنه يمكننا القول إن الجزء الديناميكي لمتطلبات العميل يتم تطبيقه باستخدام Servlet وتضمينه داخل صفحة JSP التى تحتوى على تعليمات عن كيفية عرض المخرجات عند العميل. يوضح شكل (١٠-١٤) مثالاً لاستخدام تقنية JSP لعرض قائمة الأفلام بالإضافة إلى عرض قائمة الممثلين المشاركين فى الفيلم المحدد بقائمة الأفلام.



شكل (١٠-١٤) مثال لاستخدام تقنية JSP

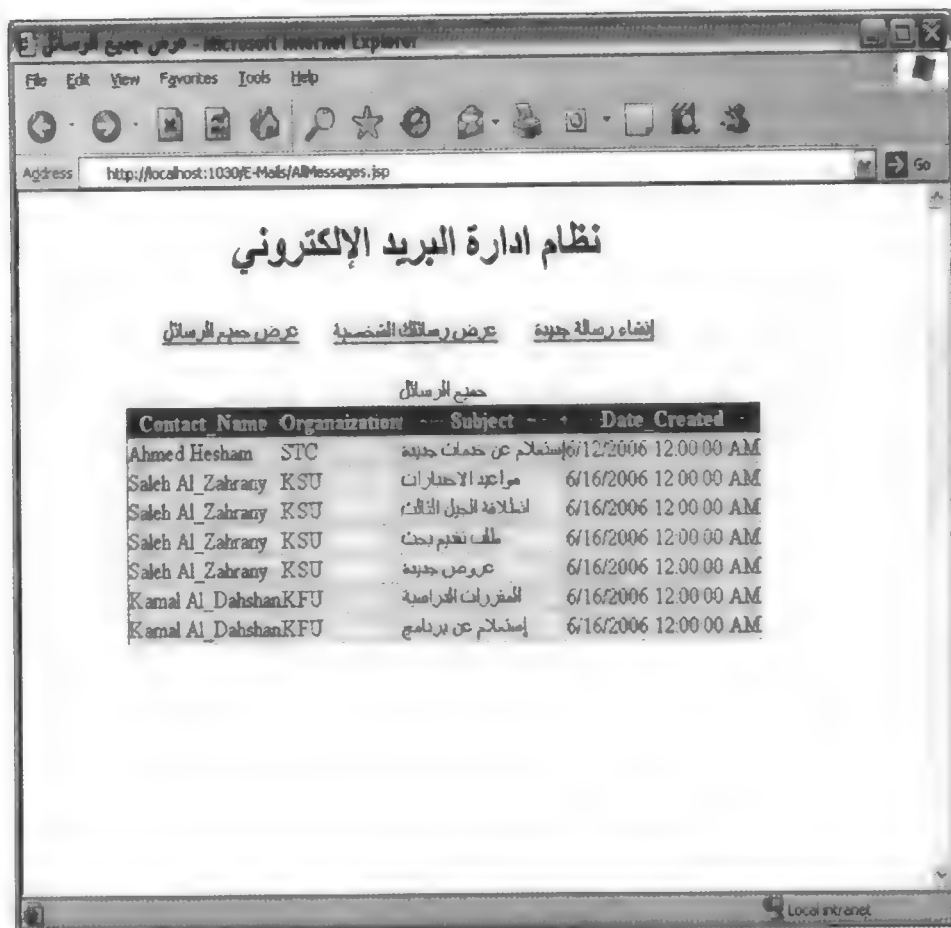
١٠-٨ تطبيق على واجهات المستخدم User Interface Application

سنحاول الآن تطبيق تقنية Servlet لتصميم وتنفيذ واجهات استخدام تطبيق إدارة البريد الإلكتروني EM، بحيث يتم تنفيذه بالكامل من خلال الويب. يبدأ التطبيق بعرض شاشة الدخول للمستخدم Login باستخدام لغة HTML كما هو موضح بالشكل (١٠-١٥). يتم استدعاء Servlet لتنفيذ عملية التحقق من بيانات المستخدم عندما يتم الضغط على مفتاح "دخول".



شكل (١٠-١٥) شاشة الدخول للتطبيق EM

وبعد نجاح عملية الدخول يتم عرض الصفحة الرئيسية للمستخدم وهي تحتوي على مجموعة الرسائل الخاصة بالمستخدم ولم يتم إرسالها كما هو موضح بالشكل (١٠-١٦). ومن خلال هذه الصفحة يمكن للمستخدم أن يستعرض محتويات أى رسالة وأن يقوم بإرسالها. جدير بالذكر أن Servlet نفسه المستخدم فى تنفيذ عملية التحقق من بيانات المستخدم هو الذى يقوم الآن بتنفيذ خدمات هذه الصفحة أيضاً.



شكل (١٠-١٦) الصفحة الرئيسية لبريد المستخدم

وعندما يقوم المستخدم باختيار إحدى الرسائل غير المرسلة ويضغط على مفتاح select فإنه يتم عرض محتويات الرسالة في الجزء العلوي من الصفحة وهو يتضمن أيضاً مفتاح Modify لإمكانية التعديل في محتوى الرسالة كما هو موضح بالشكل (١٠-١٧).

وعندما يضغط المستخدم على مفتاح Modify يتم الانتقال إلى صفحة تعديل محتوى الرسالة وهي تحتوي على جميع خصائص الرسالة ومحتوياتها الحالية مع إمكانية التعديل في أي خاصية منها كما هو موضح بالشكل (١٠-١٨). وبعد الانتهاء

من إجراء التعديل على الرسالة يتم الضغط على مفتاح Update، فيقوم Servlet بتنفيذ التعديلات المطلوبة عند خادم الويب.



شكل (١٠-١٧) عرض محتويات الرسائل

وعندما يطلب المستخدم إنشاء رسالة جديدة من خلال الصفحة الرئيسية (شكل (١٠-١٦)) فإنه يتم الانتقال إلى الصفحة الموضحة بالشكل (١٠-١٩) وهي تحتوي على جميع خصائص الرسالة، بعضها متروك خالي للتعبئة من قبل المستخدم، والبعض الآخر يحتوي على معلومات افتراضية مثل: تاريخ الإنشاء واسم المستخدم. وبعد الانتهاء من تعبئة الرسالة الجديدة يقوم المستخدم بالضغط على مفتاح save لحفظ الرسالة.

نظام البريد الإلكتروني

Edit Message No. : 1

Subject : إشعار عن ملفات جديدة

Creator Name : Ahmed Al_Zahrany

Creation Date : 12/6/2006

Assigned To : Nasser Al_Meshary

Addressed To : Saleh Al_Zahrany

Schedule Date :

Message Body : الرجاء التكرم بإرفاقنا بملفات

شكل (١٠-١٨) تعديل محتويات الرسالة

نظام البريد الإلكتروني

Message No. : 6

Subject :

Creator Name : Ahmed Al_Ghamdy

Creation Date : 6/17/2006

Assigned To : Ahmed Al_Ghamdy

Addressed To : Ahmed Hesham

Schedule Date :

Message Body :

شكل (١٠-١٩) إنشاء رسالة جديدة

المراجع

أولاً - المراجع الأجنبية:

1. Sommerville I. . "Software Engineering 7 ".7th Edition. Pearson. Addison-Wesley. Harlow. Essex. UK. 2004.
2. Pressman . "Software Engineering: A Practitioner's Approach". 5th edition. McGraw-Hill. 2001.
3. Leszek Maciaszek and Bruc Liong . "Practical Softwrae Engineering: a case study approach". Addison-Wesley. 2005.
4. J. Rumbaugh. I. Jacobson. and G. Booch . "The Unified Modeling Language Reference Manual". second edition. Addison Wesley. 2005.
5. Dwayne Phillips. "The Software Project Manager's Handbook". second edition. Wiley-Interscience. 2004.
6. Frank Tsui "Managing Software Projects". Jones and Bartlett. 2004.
7. James Taylor. "Managing Information Technology Projects". AMACOM. 2004.
8. Ghezzi. Jazayeri. and Mandrioli "Fundamentals of Software Engineering". Prentice Hall. 2003.
9. Joseph Schmuller. "Teach Yourself UML". SAMS. 2004.
10. Martin Fowler . "UML Distilled". third edition. Addison Wesley. 2004.
11. J. George. D. Batra. J. Valacich. and J. Hoffer. "Object-Oriented System Analysis and Design". Prentice Hall. 2004.
12. Dennis Lock "Project Management". eighth edition. Gower. England. 2003.
13. Sun Microsystems. "Java Programming language". Course S-110. sun Educational Services. 2003.

14. Sun Microsystems. "Java Programming language", Course SL-275. sun Educational Services. 2003
15. Eve Andersson, Philip Greenspun, and Andrew Grunt , "Software Engineering for Internet applications", MIT Press . 2003.
16. Stephen R. Schach. "Object-Oriented and Classical Software Engineering". 5th Edition. McGraw-Hill. 2002.
17. J. George. J. Valacich, and J. Hoffer. "Modern System Analysis and Design", third edition. Prentice Hall. 2002.
18. J. Hoffer, M. Prescott, and F. McFadden. "Modern Database Management", sixth edition. Prentice Hall. 2002.
19. T. Connolly, and C. Begg. "Database Systems: A Practical approach to Design, Implementation, and Management" third edition. Addison Wesley. 2002.
20. Herbert Schildt , "The Complete Reference, JAVA2", fifth edition. Osborne. 2002..
21. Deitel دة Deitel. "JAVA How to Program", 4th Edition. Prentice Hall. 2002.
22. Savitch. "JAVA: An Introduction to computer Science & Programming", Second Edition. Prentice Hall. 2002..
23. Stevens P. with Pooley, R. "Using UML: Software Engineering with Objects and Components", Addison-Wesley, Harlow, Essex, UK. 2000.
24. Gamma, Helm, Johnson, and Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software", MA. Addison-Wesley. 1995.
25. eProject(2006) , <http://www.eproject.com>.
26. eRoom(2006). <http://www.eroom.net/eroomnet/>
27. ManagePro(2006). <http://www.managepro.net/>
28. Microsoft(2006). <http://www.microsoft.com.office.project/default.asp>
29. Oracle(2006). <http://www.oracle.com/products/jdev/content.html>
30. Palisade(2006). <http://www.palisade-europe.com>

31. Primavera(2006). <http://www.primavera.com>
32. Rational(2006). <http://www.rational.com>
33. Sun(2006). J2EE platform. available at <http://www.java.sun.com>
34. Sun ONE(2006). <http://www.sun.com/software/sundev/jde/>
35. Sybase(2006). <http://www.sybase.com/products/enterprisemodeling>

ثانياً - المراجع العربية:

- ١- د. عبدالوهاب السماك، "إدارة مشاريع البرمجيات"، معهد الإدارة العامة، الرياض، ٢٠٠٥.
- ٢- د. عبدالوهاب السماك، "تخطيط وتنفيذ مشاريع مراكز الحاسب الآلي"، معهد الإدارة العامة، الرياض، ٢٠٠٤.
- ٣- د. عبدالوهاب السماك، "تحليل النظم باستخدام لغة UML"، معهد الإدارة العامة، الرياض، ٢٠٠٥.
- ٤- د. عبدالوهاب السماك، "تحليل النظم الهيكلية"، معهد الإدارة العامة، الرياض، ٢٠٠٣.
- ٥- د. عبدالوهاب السماك، "البرمجة بلغة جافا"، معهد الإدارة العامة، الرياض، ٢٠٠٤.
- ٦- محمد رسمى، "تصميم النظم باستخدام لغة UML"، معهد الإدارة العامة، الرياض، ٢٠٠٥.
- ٧- عبدالحميد بسيوني، "أساسيات هندسة البرمجيات"، دار الكتب العلمية للنشر والتوزيع، القاهرة، ٢٠٠٥.
- ٨- د. نزار الحافظ، د. ندى غنيم، د. غيداء ريداوى، "هندسة البرمجيات: منهج للممارس" مطبوعات الجمعية العلمية السورية للمعلوماتية، دمشق، ٢٠٠٢.

المؤلف فى سطور

د. عبدالوهاب كامل محمد السماك.

من مواليد القاهرة - جمهورية مصر العربية.

المؤهل العلمى:

دكتوراه الفلسفة فى هندسة الحاسبات، عام ١٩٩٢م من جامعة الزقازيق بجمهورية مصر العربية.

الوظيفة الحالية:

أستاذ الحاسبات المساعد بمعهد الإدارة العامة، الرياض.

الخبرات العلمية والعملية:

- له العديد من الأبحاث العلمية المنشورة فى المؤتمرات والدوريات العلمية المتخصصة فى هندسة الحاسبات.
- التدريس فى عدد من الجامعات المصرية (الحكومية، الأهلية).
- المشاركة فى مشروع "تطوير التعليم الهندسى" فى مجال هندسة الحاسبات بجمهورية مصر العربية.
- المشاركة فى مشروع "اللغويات الحاسوبية العربية" التابع لأكاديمية البحث العلمى بجمهورية مصر العربية.
- عضو لجنة تطوير دبلوم برمجة الحاسب فى قطاع الحاسب الآلى بمعهد الإدارة العامة.
- إعداد مجموعة من الحقائق التدريبية فى مجالات تحليل وتصميم النظم والبرمجة وإدارة مشاريع البرمجيات لقطاع الحاسب الآلى بمعهد الإدارة العامة.
- المراجعة العلمية لبعض الكتب المنشورة فى مطبوعات معهد الإدارة العامة فى مجال الحاسب الآلى.
- الإشراف على مشاريع التخرج لطلبة دبلوم برمجة الحاسب بمعهد الإدارة العامة.

حقوق الطبع والنشر محفوظة لمعهد الإدارة العامة ولا يجوز
اقتباس جزء من هذا الكتاب أو إعادة طبعه بأية صورة دون
موافقة كتابية من المعهد إلا في حالات الاقتباس القصير
بغرض النقد والتحليل، مع وجوب ذكر المصدر.

تم التصميم والإخراج الفنى والطباعة فى
الإدارة العامة للطباعة والنشر بمعهد الإدارة العامة - ١٤٢٩هـ

هذا الكتاب :

تشهد معظم البلدان العربية الآن طفرة كبيرة في استخدام تقنية المعلومات في المجالات المختلفة في محاولات جادة لتطبيق إستراتيجيات الحكومة الإلكترونية من خلال تطوير تطبيقات الأعمال الخاصة بكل مجال.

تتسم معظم تطبيقات الأعمال بـ كبير حجمها واحتوائها على قدر كبير من التعقيد، ومن ثم فهي تحتاج إلى تطوير برمجيات تتميز بالجودة والمرونة وسهولة الصيانة.

ومع ازدياد اعتماد المؤسسات والهيئات والشركات على استخدام تقنية المعلومات في إدارة وتشغيل الأنشطة الخاصة بها، فقد أدى ذلك إلى زيادة في متطلبات ومواصفات البرمجيات لكي تكون أكثر سرعة ودقة ومرونة وأمنًا وقابلة للاتساع والتحديث، بالإضافة إلى إمكانيات التعامل مع تطبيقات أخرى والعمل في بيئات تشغيل مختلفة.

إن عملية تطوير البرمجيات لتلك التطبيقات تعتبر مكلفة وتستغرق وقتًا طويلاً، ويقوم على تنفيذها فرق عمل مختلفة، لذا فإنها تحتاج إلى إدارة جيدة لكي تتمكن من تحقيق أهدافها بدقة. ويتطلب ذلك استخدام مجموعة من الأساليب والأدوات الحديثة التي تدعم عمليات تطوير البرمجيات بشكل جيد.

وسوف نركز في هذا الكتاب على أسلوب تطوير البرمجيات المبني على الكائنات ليلانم مشاريع البرمجيات الكبيرة الحجم والقابلة للاتساع. وسوف نستعين ببعض التطبيقات والحالات الدراسية لتعزيز عملية فهم كيفية استخدام تلك الأساليب.